1

RL-TR-94-150
Final Technical Report
August 1994

NEURAL NETWORK COMMUNICATIONS SIGNAL PROCESSING

Harris Corporation

Dennis Tebbe, John Doner, and Tom Billhartz

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

APPROVED: Scott Shyne

SCOTT S. SHYNE
Project Engineer

FOR THE COMMANDER: John A Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE August 1994	3. REPORT TYPE AND DATES COVERED Final Sep 92 - May 94

4. TITLE AND SUBTITLE NEURAL NETWORK COMMUNICATIONS SIGNAL PROCESSING	5. FUNDING NUMBERS C - F30602-92-C-0051 PE - 62702F PR - 4519 TA - 42 WU - 78
6. AUTHOR(S) Dennis Tebbe, John Doner, and Tom Billhartz	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Harris Corporation Government Communications Systems Division P.O. Box 91000 Melbourne FL 32902	8. PERFORMING ORGANIZATION REPORT NUMBER N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (C3BB) 525 Brooks Road Griffiss AFB NY 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-94-150

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Scott S. Shyne/C3BB/(315) 330-4819

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This final technical report describes the research and development results of the Neural Network Communications Signal Processing (NNCSP) Program. The objectives of the NNCSP program are to: (1) develop and implement a neural network and communications signal processing simulation system for the purpose of exploring the applicability of neural network technology to communications signal processing; (2) demonstrate several configurations of the simulation to illustrate the system's ability to model many types of neural network based communications systems; and (3) use the simulation to identify the neural network configurations to be included in the conceptual design of a neural network transceiver that could be implemented in a follow-on program.

14. SUBJECT TERMS Neural networks, Signal processing, Communications	15. NUMBER OF PAGES 130
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

1. INTRODUCTION

This Final Technical Report describes the research and development results of the Neural Network Communications Signal Processing (NNCSP) Program, contract number F30602-92-C-0051. The objectives of the NNCSP Program are to: 1) develop and implement a neural network and communications signal processing simulation system for the purpose of exploring the applicability of neural network technology to communications signal processing, 2) demonstrate several configurations of the simulation to illustrate the system's ability to model many types of neural network based communication systems, and 3) use the simulation to identify the neural network configurations to be included in the conceptual design of a neural network transceiver that will be developed in a phase II follow-on program.

1.1 BACKGROUND

Possible application areas for neural network technology in the communication system domain are the signal processing functions of transceivers which include noise cancellation, demodulation, decoding and channel equalization. A modular design approach that couples neural network modules with conventional signal processing modules has the potential of producing a smart radio that exhibits a flexible design and enhances link survivability in an electronically hostile environment. The current status of neural networks and their application to the communication system domain remains in a state of basic research. This basic research has resulted in several papers with general theory and some preliminary results but very little else. These papers provide an initial baseline and provide motivation for obtaining a computer aided design system for the creation of neural network based communication systems. Ongoing work for the "Speakeasy" multiband, multimode radio program and the smart radio development program at Rome Laboratory requires support in the application areas mentioned above. Neural networks may be able to provide communication systems with a great deal of processing power while at the same time providing a degree of fault tolerance. Previous efforts have addressed small aspects of communication systems using neural network technology but an overall simulation system that compares neural network techniques with conventional signal processing techniques is unique to this effort. The successful completion of this effort significantly enhances the state of the art in design capability for smart radio technology.

1.2 SCOPE

This program represents the first of two developmental phases. In the first phase, the contractor designs and implements a software simulation environment that will be used for modeling neural network based communications systems. In the second phase, the contractor will fabricate a breadboard transceiver based on the results of the phase I effort. This contract only addresses the phase I neural network communications signal processing simulation, and the remainder of this description of the technical scope addresses only the tasks of phase I.

The first task of the NNCSP program was a feasibility study to determine which Neural Network paradigms can best be applied to the communication domain. The results of this feasibility study formed the basis for specifying the neural network modules in the software simulation environment. The next task was the design and development of a software package which provides the capability for communication engineers to design and test communication systems that contain modules of neural network algorithms and also modules of conventional signal processing techniques. The simulation software provides the capability to interchange neural network modules with similar conventional signal processing modules. When the simulation software was completed, the next task was to demonstrate several configurations of the simulation to illustrate the system's ability to model many types of neural network based communication systems. The following task consisted of a series of simulation experiments aimed at identifying neural network based communications signal processing functions which should be included in future neural network based transceivers. As part of phase I, the last task was a high-level conceptual design of a prototype neural network communication system which is recommended for a phase II implementation . Based on its review of phase I, the Government will decide whether to proceed with a procurement of a phase II effort.

The software package that was developed and implemented in this contract is named the Neural Network Communications Simulation System (NNCSS). It is a communications-oriented digital signal processing (DSP) simulator with the capability to invoke neural network paradigms into signal processing chains. This is a generic tool which will greatly facilitate the design and simulation of communication signal processing products that incorporate embedded neural network technology.

The NNCSS is based on the Signal Processing WorkSystem™ (SPW™), a commercial off the shelf (COTS) signal processing simulator, offered by Comdisco Systems, Inc. SPW provides a block diagram approach to constructing signal processing simulations. The entire prototyping effort is viewed in schematic form on the workstation monitor, providing a familiar engineering-oriented paradigm for the analyst. Using SPW, the analyst attaches special function signal processing modules to each other, and provides input signal sources, which may be either digital data retrieved from disk or real-time analog signals digitized "on the fly" as the simulation runs. The analyst has the capability to insert "probes" at various points in the signal processing chain and view signal characteristics at the probed points. Such probes may collect data for real-time viewing or post simulation analysis.

In this contract, additional function blocks and codes have been added to SPW to allow the design and simulation of neural network based communication functions. With the functions provided by the Neural Network Communications Library, the NNCSS allows the analyst to compare the performance achieved by neural network based designs with the performance of similar functions which use conventional signal processing approaches.

1.3 REPORT ORGANIZATION

Section 2 presents the NNCSS. Specifically, the feasibility study which defined the NNCSS is summarized in 2.1,and the NNCSS architecture is described in 2.2. Section 3 describes the Neural Network Communications Library (NNCL) which is the library of neural network function blocks used within the NNCSS environment. Section 4 describes all of the simulation experiments which were used to investigate the application of neural networks in communications signal processing. Section 5 presents the high-level conceptual design of a neural network transceiver which could be developed in a phase-II follow-on program,. Section 6 presents the conclusions of the program and recommendations for additional work to be done.

1.4 REFERENCE DOCUMENTS

1.4.1 Government Documents

The following documents define the contractual requirements for the Neural Network Communications Signal Processing Program

DOD-STD-2167A, Defense System Software Development, 29 February 1988.

DI-MISC-80711/T, Data Item Description (DID).for Scientific and Technical Reports (Final).

F30602-92-R-0005, Request for Proposal, Neural Net Communications Signal Processing, Rome Laboratory, Griffiss Air Force Base.

F30602-92-C-0051, Contract, Neural Network Communications Signal Processing Program, Rome Laboratory, Griffiss Air Force Base.

1.4.2 Non-government Documents

The following documents are contractual data requirements of the Neural Network Communications Signal Processing Program:

Feasibility Study, Technical Information Report for the Neural Network Communications Signal Processing Program, CDRL A003, 31 March 1993.

Software Development Plan for the Neural Network Communications Signal Processing Program, CDRL A005, 1 March 1993.

System/Segment Design Document for the Neural Network Communications Simulation System, 6 June 1994.

Software Design Document for the Neural Network Communications Library, CDRL A004, 6 June 1994.

Software Design Document for the Neural Network Object Manager, CDRL A004, 6 June 1994.

Software Test Description for the Neural Network Communications Simulation System, CDRL A006, 6 June 1994.

Software Test Report for the Neural Network Communications Simulation System, CDRL A007, 6 June 1994.

Software Users Manual for the Neural Network Communications Simulation System, CDRL A008, 7 June 1994.

The following documents define the Signal Processing WorkSystem™ that comprises the commercial off the shelf (COTS) computer software configuration items (CSCI) that are part of the Neural Network Communications Simulation System:

SPW™ - The DSP Framework™ User's Guide and Tutorial, Product Number: SPW8010, Document Version: 3.0, Comdisco Systems, Inc., September 1992.

SPW™ - The DSP Framework™ Macro Command Language Reference, Product Number: SPW8011, Document Version: 3.0, Comdisco Systems, Inc., September 1992.

SPW™ - The DSP Framework™ Designer/BDE™ User's Guide, Product Number: SPW8012, Document Version: 3.0, Comdisco Systems, Inc., September 1992.

SPW™ - The DSP Framework™ Signal Calculator™ User's Guide, Product Number: SPW8013, Document Version: 3.0, Comdisco Systems, Inc., September 1992.

SPW™ - The DSP Framework™ Signal Flow Simulation User's Guide, Product Number: SPW8014, Document Version: 3.0, Comdisco Systems, Inc., September 1992.

SPW™ - The DSP Framework™ DSP & BOSS™ Communications Library Reference, Product Number: SPW8015, Document Version: 3.0, Comdisco Systems, Inc., September 1992.

SPW™ - The DSP Framework™ Tool Interface Language Reference, Product Number: SPW8016, Document Version: 3.0, Comdisco Systems, Inc., September 1992.

SPW™ - The DSP Framework™ Standard C Code Generation System™, Product Number: CGS8000-C, Document Version: 1.6, Comdisco Systems, Inc., September 1992.

SPW™ - The DSP Framework™ Interactive Simulation Library™ Reference, Product Number: ISL8000, Document Version: 1.0, Comdisco Systems, Inc., September 1992.

2. NEURAL NETWORK COMMUNICATIONS SIMULATION SYSTEM

The development of the NNCSS provides a communications-oriented digital signal processing (DSP) simulator which smoothly incorporates the capability to invoke neural network paradigms into signal processing chains. This is intended to be a generic tool which will greatly facilitate the design and simulation of communications signal processing products that incorporate embedded neural network technology.

The NNCSS is based on the Signal Processing WorkSystem™ (SPW™), a commercial off the shelf (COTS) signal processing simulator, offered by Comdisco Systems, Inc. SPW provides a block diagram approach to constructing signal processing configurations for simulation.

The users of NNCSS are expected to be communication analysts and designers with a detailed understanding of communication signal processing and only an elementary understanding of neural network techniques. The NNCSS will cast the neural network paradigms into a function block form that will allow the analyst to design and simulate neural network based communication functions in a manner similar to what is currently done using conventional design approaches.

The primary mission of the NNCSS will be to support investigations in the design of specific communication systems based in part or totally on neural network approaches. The NNCSS will allow comparison of neural network and conventional communication system designs.

2.1 FEASIBILITY STUDY

The first task in the NNCSP Program has a feasibility study which identified a feasible approach for developing a simulation software package for neural network communications signal processing. The feasibility study also selected a set of neural network paradigms which could provide a comprehensive set of neural network capabilities. That part of the feasibility study which dealt with the selection of neural network paradigms is summarized in 2.1. The software architecture is summarized in 2.2.

2.1.1 Neural Network Applications in Communications

A literature search was undertaken as a precursor to the initiation of NNCSS software development. This search focused on determining what sorts of neural network paradigms have been successfully applied to communications signal processing problems. The search included the DIALOG database search system, and a search of the Defense Technical Information Center (DTIC). Several papers also were found in IEEE periodicals, including papers from IEEE Transactions on Communications.

A search was performed using the key words "Neural()Network" and "Communication()System" on the DIALOG database which contains articles from published technical periodicals. This database produced 98 matches. Of the titles observed, there were 17 which appeared to be very relevant. Abstracts for these 17 were ordered, and the full text of 13 were acquired.

A search was also performed on the DTIC using the key words "neural nets" and "communication and radio systems". DTIC has a specific guide which constrained the key word choices. Nineteen documents were discovered by this search; six were of interest and were acquired.

In summary, the literature search produced a total of 46 papers (included in the Bibliography, Section 7.0) which appeared relevant to NNCSS. These 46 papers were individually reviewed, with a short report being generated for each paper which in fact had strong relevance to the expected applicability of the NNCSS. Of the original 46 papers, 19 were of direct interest. These 19 are identified in the following list.

List of Applicable Papers

1. Aazhang, B., et al., "Neural Networks for Multiuser Detection in CodeDivision Multiple-Access Communications"
2. Anderson, J., et al., "Radar Signal Categorization Using a Neural Network"
3. Andersson, G., et al., "Generation of Soft Information in a Frequency-Hopping HF Radio System Using Neural Networks"
4. Chesmore, E.D., "Application of Pulse Processing Neural Networks in Communications and Signal Demodulation"
5. Feiz, S., et al., "Adaptive ML Neural Network Based Receiver for Q^2PSK Modulated Data Transmission Systems"
6. Fontana, R., et al., "Communications Signal Recognition and Demodulation via Neural Networks"
7. Hussain, M., et al., "Neural Network Application to Error Control Coding"
8. Kohonen, T., et al., "Combining Linear Equalization and Self-Organizing Adaptation in Dynamic Discrete Signal Detection"
9. Jeffries, C., "High Order Neural Models for Error Correcting Code"
10. Johnson, J. , "Neural Network Algorithm Decoding and Sequence Predictor"
11. Kechriotis, G., et al., "Using Recurrent Neural Networks for Blind Equalization of Linear and Nonlinear Communication Channels"
12. Lee, T., et al., "Adaptive Vector Quantization Using a Self-Development Neural Network"
13. Naylor, J., "A Neural Network Algorithm for Enhancing Delta Modulation/LPC Tandem Connections"
14. Santamaria, M., et al., "Neural Net Filters: Integrated Coding and Signalling in Communication Systems"
15. Rao, S., et al., " A Neural Network Tunable Filter for Multi-Tone Detection"
16. Siu, S., et al., "Decision Feedback Equalization Using Neural Network Structures"
17. Spect, D., "Probabilistic Neural Networks"
18. Tasic, J., et al., " Theory and Application of the Neural Net-Based Adaptive Filter in Communication Systems"
19. de Veciana, G., et al., "Neural Net-Based Continuous Phase Modulation Receivers"

2.1.2 Neural Network Paradigm Choices for NNCSS

This section summarizes the evaluation of the applicable papers and the selection of neural network paradigms for NNCSS.

Table 2-1 summarizes the 19 papers mentioned above. Note that the uses of neural paradigms comprised:

1. use of backpropagation networks in nine applications,

2. use of associative recurrent networks in four applications, specifically Hopfield and Brain State in a Box (BSB),

3. use of Kohonen feature map structures in two applications,

4. use of ART once,

6. use of the Probabilistic Neural Network once, and

5. use of two rather unique networks in one application each, specifically Time Discriminant (TD) and SPAN.

The types of communications applications to which these networks were applied can be taxonomized as

1. channel equalization,

2. interference rejection,

3. signal detection,

4. multipath rejection,

5. baseband data recognition and/or compression,

6. error detection and correction,

7. signal source identification, and

8. digital sequence prediction.

Table 2 - 1 Summary of Applications Relevant to NNCSS

APPLICATION	NETWORK PARADIGM	TRAINING PROCEDURE	REFERENCE NUMBER
CDMA signal detection	two-layer perceptron	backpropagation	1
Signal source identification	brain state in a box (BSB)	dynamic feedback	2
Error detection and correction	two-layer perceptron	back propagation, continuous learning	3
Signal detection	TD network (unconventional)	hardwired	4
Q^2PSK signal detection	Hopfield/Tank	hardwired	5
16 QAM, 8-PSK signal detection	variant of ART	continuous self-organizing	6
error detection and correction	three-layer perceptron	backpropagation	7
nonlinear channel equalization	Self-Organizing Feature Map	autoadaptive	8
error detection and correction	associative content addressable memory	hardwired	9
pseudorandom sequence prediction	three-layer perceptron	backpropagation	10

APPLICATION	NETWORK PARADIGM	TRAINING PROCEDURE	REFERENCE NUMBER
nonlinear channel equalization	recurrent network	real time recurrent learning	11
data compression	SPAN (unconventional)	autoadaptive	12
channel equalization	Self-organizing feature map	autoadaptive	13
Adaptive filtering	multilayer perceptron	backpropagation	14
signal detection	three layer perceptron	backpropagation	15
intersymbol interference	three layer perceptron	backpropagation	16
signal source identification	probabilistic neural network	computed from training vectors	17
multipath correction	three-layer complex perceptron	backpropagation	18
CPM signal detection	two-layer perceptron	backpropagation	19

It should be noted that some of the neural applications simultaneously represented applications in more than one area, especially the application papers dealing with signal detection/channel equalization schemes jointly carried out in a single neural architecture.

Of these applications, the highly successful ones tended to be those which replaced computationally intense processes of traditional signal processing. In particular, the applications in signal detection/channel equalization and error correction, where Viterbi processes (normally quite computationally complex) were replaced or augmented tended to provide large performance gains, and all of the forms of signal detection tended to be good or adequate, although some of them were not as good as (computationally complex) optimal Bayesian decision rules. Another strong point of several applications dealing with channel impairment was the unique ability of the neural paradigms to adapt to varying signal environments. All of these papers documented work in which the neural application was found to work as well as conventional approaches, at least over some practical range of operation.

The application types and network types found in the literature search elucidate the fact that many neural network paradigms are useful in signal processing applications, and in fact, different neural paradigms were successfully applied to the same application area, as illustrated in Table 2-1. It would be desirable for the NNCSS to support a range of neural paradigms which provide the capabilities, in terms of architectures, training techniques, and mathematical estimation possibilities, that are found in the neural literature. At the same time, it is best if the NNCSS can deliver that range of versatility with a small collection of well-known and well-documented paradigms.

Following this approach, the paradigms to be included were determined by creating a prioritized list of criteria, and finding a set of networks which fulfilled all of those criteria. The neural paradigms were chosen by satisfying the highest priority criterion not yet met by prior selections with the best unrepresented neural paradigm meeting that criterion. Of course, in some cases, a single paradigm satisfies several criteria, and in most cases, several paradigms satisfy individual criteria. In this way a relatively small collection of neural paradigms provides a wide range of neural capabilities to the NNCSS user.

Table 2-2 below provides a synopsis of the neural paradigm choice process. The rows of the table are indexed by the criteria of choice, ordered in importance, with the topmost row reflecting the criterion deemed of greatest importance. The columns of the table reflect the neural paradigms chosen to satisfy these criteria, and the marked row-column intersections show which criteria are satisfied by each paradigm.

Table 2-2 Choices of Neural Paradigms for the NNCSS

#	Criterion	Backpropagation	Kohonen SOFM	Recurrent Backprop	ART 2	BSB	Probabilis-tic
1	Most popular	X					
2	Backpropagation	X					
3	Supervised learning	X		X		X	X
4	Unsupervised learning		X		X	X	X
5	Recurrent			X			
6	Variable topology				X		
7	Associative					X	
8	Competitive learning		X		X		
9	One pass training						X
10	Applied	X	X	X	X	X	X

The criteria of table 2-2 are elaborated as follows:

1. the most popular architecture found in applications is the multilayer perceptron structure trained using backpropagation;

2. the backpropagation learning algorithm is known to have important theoretical properties and has performed well in many previously documented applications;

3. several network architectures can be trained using supervised training;

4. likewise, several autoadaptive networks, most notably the Kohonen Self-Organizing Feature Map (SOFM), can be trained in unsupervised mode;

5. recurrent networks are valuable in modeling processes with indefinite or infinite temporal memory, such as the infinite impulse response filter, and the recurrent backpropagation network is useful in such applications;

6. networks with variable topology permit the addition of new nodes in order to accommodate recognition problems where an unknown (or varying) number of pattern exemplars are involved;

7. associative networks are good at restoring partial patterns, or patterns corrupted with noise, and the Brain State in a Box is a versatile but fairly simple architecture of this type;

8. the competitive learning law (e.g., in the SOFM) can create a network which provides equiprobable exemplars relative to the probability density of the input space, and is thus a valuable technique in analysis and estimation;

9. one pass training refers to network structures that can be trained with a single pass through the data (or, in some cases, by computing the weights directly from the known probability of the function to be modeled), providing for very rapidly trained networks;

10. the networks found in the applications literature of this study happen to coincide with the choices made by the preceding nine criteria, providing a validity check that using such criteria leads to a range of paradigms which will be found in practice.

Thus, the networks shown in Table 2-2 were, at the conclusion of the feasibility study, those recommended for initial inclusion in the NNCSS, and would provide the required versatility to address a wide range of application areas.

During the preliminary design of the NNCSS it was realized that the synthesis of neural networks within a flexible signal processing simulation system, such as SPW, provided opportunities and capabilities that had not been envisioned in neural network simulation tools previously. With the NNCSS, different neural network paradigms can be combined into more complex neural network designs to create entirely different neural network solutions. For example, the reset logic of Adaptive Resonance Theory (ART) can be used to trigger state changes to a recurrent network that is mapping the features extracted by a Kohonen network to the input of a multilayered backpropagation network that is performing process control that is effecting the original input pattern. Therefore, the development of NNCSS leads to the identification and implementation of neural network paradigms that can be configured with other paradigms to form multilayered architectures. Because of this natural progression, the Adaptive Resonance Theory I (ART1) and Grossberg Outstar paradigms were added to the NNCSS.

ART1 is the original binary version of Adaptive Resonance Theory for processing binary (0,1) input patterns. This network has proven itself as a valuable network in data compression and pattern recognition. More importantly, in the NNCSS, the network provides another view of pattern features that can be used to stabilize the learning in other neural networks such as the recurrent and backpropagation networks. Likewise the Grossberg Outstar paradigm provides a simple but general technique for retrieving target patterns from self organizing networks such as the Kohonen or ART networks. For example, Hecht-Nielson's Counterpropagation network combines two outstar layers with a Kohonen layer to retrieve target patterns in both directions (input to output and output to input mappings). Also the Fully Recurrent Network, which typically is configured as a single layer network, has been modified to output a backpropagating residual error that allows the recurrent layer to feedback to other recurrent or backprop layers in a multilayered architecture.

Before any software development efffort had been expended on the Probabilistic Network (PN), consideration was given to trading the PN for the Adaptive Resonance Theory III (ART3) network. The primary advantage of the PN paradigm over other neural networks is that the learning is comparatively fast, in that the network does not need more than one pass through the training data to reach its full capability. However, the Kohonen Topological Feature Map as currently implemented has proven to be a feature extractor that converges quickly to a stable feature mapping which can then be updated on a real-time basis. Furthermore, the PN paradigm operates on a finite database of all previous patterns and does not have the same capabilities of abstraction as the Kohonen network.

It was finally decided that the software effort would be better spent on ART3 than on the Probabilistic Network because the ART3 network paradigm is strongly suggested for the following reasons:

a. Adaptive Resonance Theory III is the latest update to the Adaptive Resonance Theory and completes the trilogy of network paradigms developed by Carpenter and Grossberg.

b. ART3 is a hierarchical version of ART2 allowing for multilayered ART networks which continues the current direction of development for NNCSS, and extends the applicability of ART in communication designs.

c. The Air Force has provided the major research funding for the development of Adaptive Resonance Theory, and it seems appropriate to include ART III in an Air Force developed neural network simulation.

The final list of neural network paradigms that are implemented in the NNCSS are:

1. Backpropagation ,

2. Kohonen Feature Map and Outstar,

3. Fully Recurrent,

4. Adaptive Resonance Theory I, II, & III, and

5. Brain State in a Box.

2.1.3 Candidate Applications for NNCSS

The NNCSS Study includes effort to create several prototype applications on the NNCSS which integrate the use of neural paradigms into signal processing algorithms which are embedded in communications systems. The effort focuses on demonstrating the capabilities of the NNCSS, rather than on producing new research results in neural network theory. Thus in selecting potential applications, there were several objectives:

1. the applications should provide tutorial support to NNCSS users;

2. the theoretical structures of the applications should not be too complex, with low technical risk of implementation;

3. the applications should represent more common types of communications signal processing functions which have been implemented with neural networks;

4. the inputs required for the application should be easily obtained or easily simulated using the SPW foundation of NNCSS.

In order to illustrate the potential range of applications in communications signal processing, we reiterate the types of applications which were found in the literature, as referenced in Table 2-1; the number of occurrences of these applications are shown in Table 2-3. For completeness, this table includes one category which would seem to be a potential application area (phased array antenna control), but for which no papers were found.

The tabulation suggests that the most common applications are signal detection, channel equalization, and error detection and correction. However, the remaining categories also illustrate that useful adaptive implementations can be found for signal processing tasks both at the digital and analog signal processing levels in communications chains.

Table 2-3 Neural Network Applications in Communications Systems

Application area	paper number	number of occurrences
1. signal detection	1, 4, 5, 6, 15, 19	6
2. channel equalization	6, 8, 11, 13, 14, 16	6
3. error detection and correction	3, 7, 9	3
4. signal source identification	2, 17	2
5. interference rejection	1, 14	2
6. data compression	12	1
7. digital sequence prediction	10	1
8. multipath rejection	18	1
9. phased array antenna control	none	0

The results of the literature search, together with the objectives set forth at the beginning of this section, and a subjective judgment of the value of neural networks (as compared to conventional technology) in different application areas leads to selection of

1. channel equalization,
2. interference rejection,
3. signal detection,
4. multipath rejection and combining,
5. baseband data recognition and/or compression,
6. error detection and correction, and
7. signal source identification,

as those applications areas which were targeted for further investigation using the NNCSS. This further investigation also was focused on the advantages of neural architectures in adaptive situations.

2.2 SYSTEM ARCHITECTURE

The NNCSS was developed by integrating SAIC's Neural Network Object Manager (NNOM) and Industrial Strength Neural Networks (ISNN) Library into Comdisco's Signal Processing WorkSystem™ (SPW™). SPW provides interfaces and tools for integrating custom function blocks. The NNCSS development involves developing neural network function blocks and supporting tools that interact with ISNN via the NNOM.

Figure 2-1 gives the overall architecture and illustrates the development activity. The square rectangles indicate non-development items based upon already existing codes. The rounded rectangles identify development activities required to integrate neural network function blocks to SPW. The arrows indicate the dependencies among the configuration items and the development activities required to integrate function blocks into SPW.

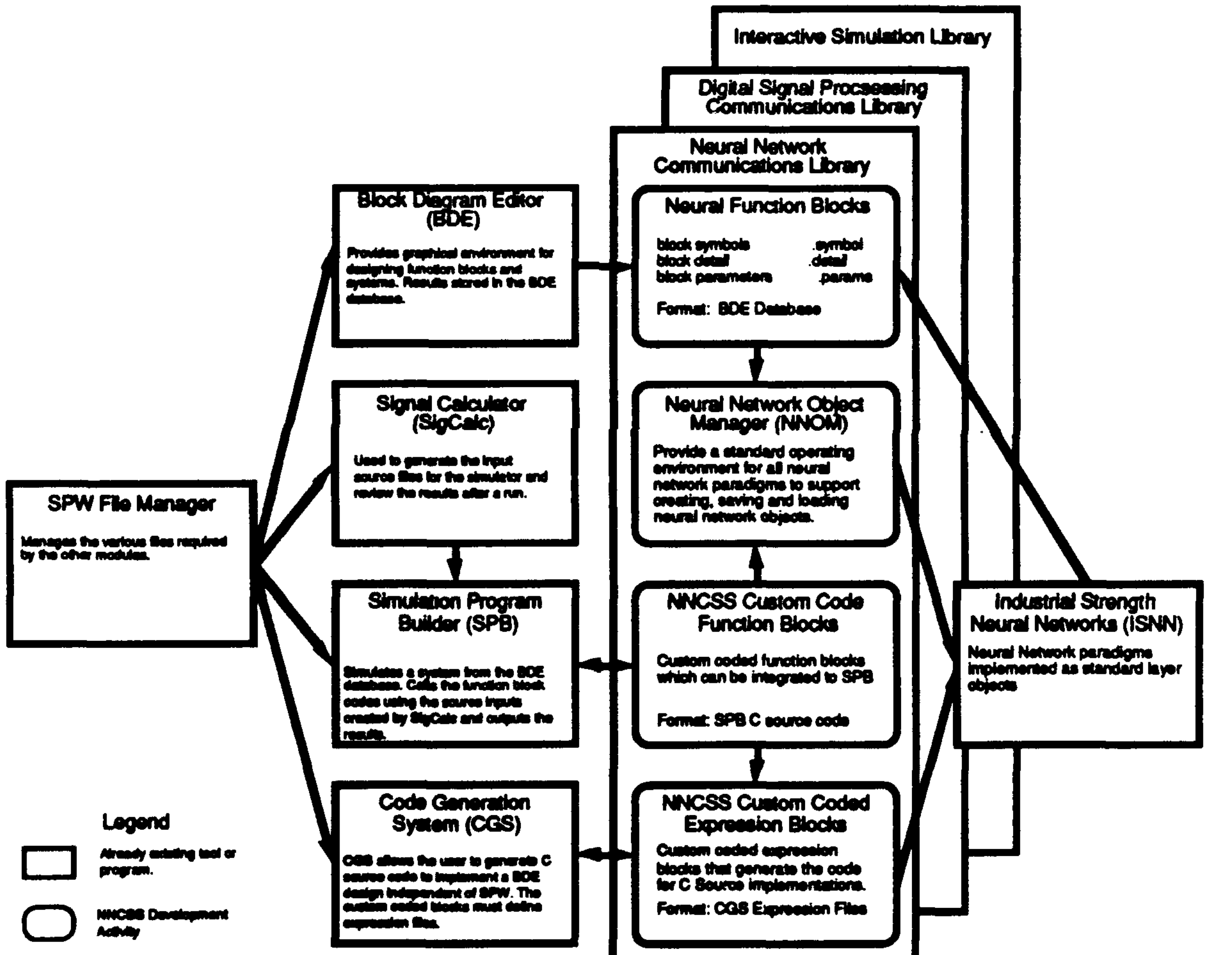

Interactive Simulation Library
Digital Signal Processing Communications Library
Neural Network Communications Library
Block Diagram Editor (BDE)
Provides graphical environment for designing function blocks and systems. Results stored in the BDE database.
Signal Calculator (SigCalc)
Used to generate the input source files for the simulator and review the results after a run.
SPW File Manager
Manages the various files required by the other modules.
Simulation Program Builder (SPB)
Simulates a system from the BDE database. Calls the function block codes using the source inputs created by SigCalc and outputs the results.
Code Generation System (CGS)
CGS allows the user to generate C source code to implement a BDE design independent of SPW. The custom coded blocks must define expression files.
Legend
Already existing tool or program.
NNCSS Development Activity
Neural Function Blocks
block symbols .symbol
block detail .detail
block parameters .params
Format: BDE Database
Neural Network Object Manager (NNOM)
Provide a standard operating environment for all neural network paradigms to support creating, saving and loading neural network objects.
NNCSS Custom Code Function Blocks
Custom coded function blocks which can be integrated to SPB
Format: SPB C source code
NNCSS Custom Coded Expression Blocks
Custom coded expression blocks that generate the code for C Source implementations.
Format: CGS Expression Files
Industrial Strength Neural Networks (ISNN)
Neural Network paradigms implemented as standard layer objects

Figure 2-1 NNCSS System Architecture

Table 2-4 describes the NNCSS configuration items included in this system architecture, including commercial off the shelf (COTS) and non-development items (NDI). The Neural Network Communications Library (NNCL) Computer Software Configuration Item (CSCI) is the product of the software development effort in this contract. The software design was documented in the Software Design Document (SDD) for the NNCL CSCI.

The non-development items (NDIs) are documented in separate documentation and user manuals prepared for the item outside of this contract.

Table 2-4 NNCSS Computer Software Configuration Items

CSCI	COTS	NDI	DESCRIPTION
SFM	X	X	SPW File Management - manages the various files required by the other modules and allows the user to enter the other CSCIs.
BDE	X	X	Block Design Editor - Provides a graphical environment for designing function blocks and systems with results stored in the BDE database, which can be accessed by other CSCIs.
SigCalc	X	X	Signal Calculator - Used to generate the input source files for the simulator and review the results after a run.
SPB	X	X	Simulation Program Builder - Builds an interactive simulation program from the BDE database and runs the simulation. Calls the function block codes using the source inputs created by SigCalc and outputs the results.
CGS	X	X	Code Generator System (OPTIONAL) - Allows the user to generate C source code to implement a BDE design independent of SPW. The custom coded expression blocks must be defined for each SPB function block.
DSPCL	X	X	Digital Signal Processing Communications Library - provides function block symbols and implementations that can be used to design conventional communications systems.
ISL	X	X	Interactive Simulation Library (OPTIONAL) - Provides interactive graphical elements that can be included in a design to allow the user to interact with the simulation during a run.
NNCL			Neural Network Communications Library - Provides custom function blocks for embedding neural network functions within conventional communication designs.
NNOM		X	Neural Network Object Manager - Provides a standard operating environment for all neural network paradigms to support creating, initializing, saving, loading and deleting neural network objects within a simulation.

3. NEURAL NETWORK COMMUNICATIONS LIBRARY

Section ? summarizes the Neural Network Communications Library (NNCL). The NNCSS in operation is simply SPW with the specific inclusion of the NNCL. The information and instructions necessary for user interaction with SPW is presented in the SPW references given in Section 1.4.2. The user of the NNCSS should first study and refer to the relevant SPW manuals regarding the operation of SPW and the generic use of optional function block libraries such as NNCL. The Software User's Manual for the NNCSS presents the information about the NNCL and its function blocks that should be available to the SPW user who intends to create block diagrams with NNCL functional blocks and to run simulations that include NNCL functional blocks. This section consists of excerpts from the Software User's Manual.

3.1 NNCL OVERVIEW

The Neural Network Communications Library (NNCL) is a set of neural network function blocks that can be used with BDE to design communication systems that include neural network based signal processing functions. It implements selected neural network paradigms and provides support functions for training and processing neural networks within communication designs. It also provides special handling functions for data preprocessing and propagation of vector activations. With the NNCL, the analyst can construct various configurations of neural networks and conventional signal processing networks to perform signal processing functions.

Artificial Neural Systems represent a rather diverse set of approaches to solving problems in pattern recognition, image analysis, associative memory, classification, filtering, and prediction. The various approaches are referred to as *paradigms*. The paradigms provide the general rules and procedures for constructing a neural network to perform a specific function. The common elements of an ANS paradigm are the *processing elements* which are local centers of computation which represent an *artificial neuron*. The processing elements are connected to form a *network* with each processing element receiving input signals from other processing elements in the network and generating an output signal which propagates to other processing elements in the network. Often the processing elements are grouped into *layers* within a network with the outputs of the processing elements of one layer being distributed to the inputs to the processing elements in the next layer. The connections among the processing elements are weighted such that the output *activation signal* from the processing element has either an *excitatory* or *inhibitory* effect on the other processing elements.

The *weights* represent the memory of the system, and are formed through a process called *training*. During training, examples are presented to the network at input processing elements that initiate the propagation of signals through the network. The weights are adjusted to represent the mapping of the input pattern to an output pattern. The paradigm method for adjusting the weights is called the paradigm's *learning algorithm*. For *unsupervised* learning, the input patterns are organized internally to form categories. Such algorithm's are *self-organizing* since they form their own output patterns for each input pattern. For *supervised* learning, the network is given the desired target for a given input pattern. The network is trained to map input patterns to desired output patterns. Once trained the neural network is able to retrieve the desired output pattern for a given input pattern even if the input pattern does not exactly match any of the original training patterns. The important characteristic of the neural network is its ability to generalize from the training examples.

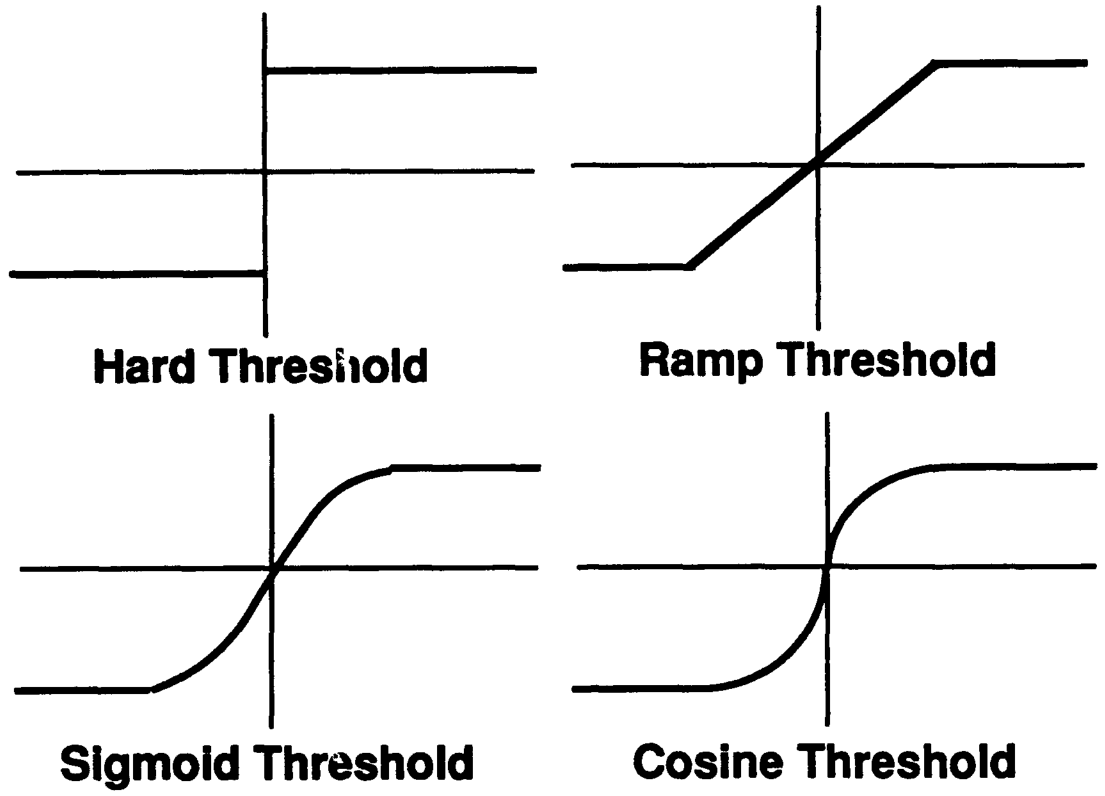


Figure 3-1 Typical Non-linear threshold functions used in ANS

Another important characteristic of neural networks is that the mapping from input patterns to output patterns is non-linear. This is achieved by passing the output signal from the processing element through a non-linear threshold function such as a ramp, hard threshold or sigmoid function (see Figure 3-1). This prevents a multi-layered network of processing elements (where the output from one set of processing elements propagates to the next layer of processing elements) from being reduced to a linear matrix operation.

3.1.1 NNCL CSCI Architecture

The basic architectural unit of the NNCL is called a *function block*. Each function block is identified as a Computer Software Unit (CSU) in the NNCL design. A function block performs a function within a signal processing network. The NNCL provides custom function blocks for incorporating neural network functions into communications system designs. There are two kinds of function blocks in the NNCL:

a. *Custom Coded Function Blocks* (CCFBs) - low level function blocks that require code to implement the function.

b. *Custom User Function Blocks* (CUFBs) - hierarchical function blocks that are composed of a connected network of low level function blocks. These do not require any custom code and can be configured and saved by the user.

The NNCL includes both low level CCFBs and higher level CUFBs that provide common configurations of CCFBs to aid the user in incorporating neural network designs into a system.

The NNCL function blocks are logically ordered so that the user can select the appropriate function block to incorporate into a communications system design. The logical groupings and sub-groupings will be identified as Computer Software Components (CSCs) of the NNCL CSCI. Table 3-1 describes the first level of groupings in the NNCL.

Table 3-1 NNCL Computer Software Components and Groupings

CSC	Grouping	Description
01	nnom	Neural Network Object Manager
02	manage	Neural Network Management
03	registers	Register Function Blocks
04	probes	Neural Network Instruments and Probes
05	backprop	Backpropagation Function Blocks
06	kohonen	Kohonen Feature Map Function Blocks
07	recurrent	Fully Recurrent Network Function Blocks
08	art	Adaptive Resonance Theory Function Blocks
09	bsb	Brain State in a Box Function Blocks

3.1.2 System States And Modes

A function block has various representations in the NNCSS depending upon the module or tool accessing the CSU. Each representation is referred to as a model and has an associated file containing the information for the model A model of a function block will be interpreted as a mode of a CSU within the NNCSS system. The specification of all of the models for a function block will provide the detailed description for each CSU in paragraph 3.3. The following paragraphs define each of the models and required content.

3.1.2.1 Symbol Model.

The symbol model defines how a function block is represented in a system's block diagram. It consists of a function block symbol (typically a block) with connectors defining the input , output, and control signals to the block (see Figure 3-2). By convention, the input connectors are to the left, and output connectors are to the right of the function block symbol. The connectors from the bottom indicate Boolean control parameters that can vary during the simulation.

The parameters shown in the function block identify the configuration properties of the function block, which distinguish the specific instance of the block. All of the parameters of a function can be shown in a separate detailed or parameter view that is linked to the symbol. The user can double click the symbol to access these views depending upon whether the block is CCFB or CUFB. The parameters can be edited to define a specific instance of the function block.

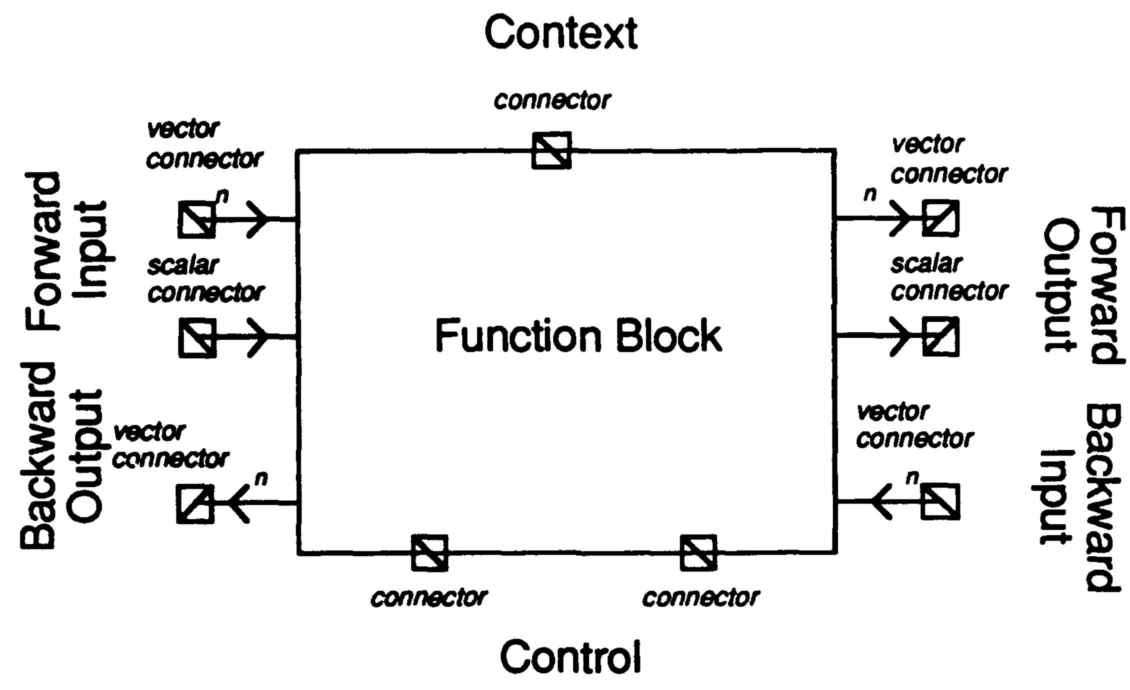
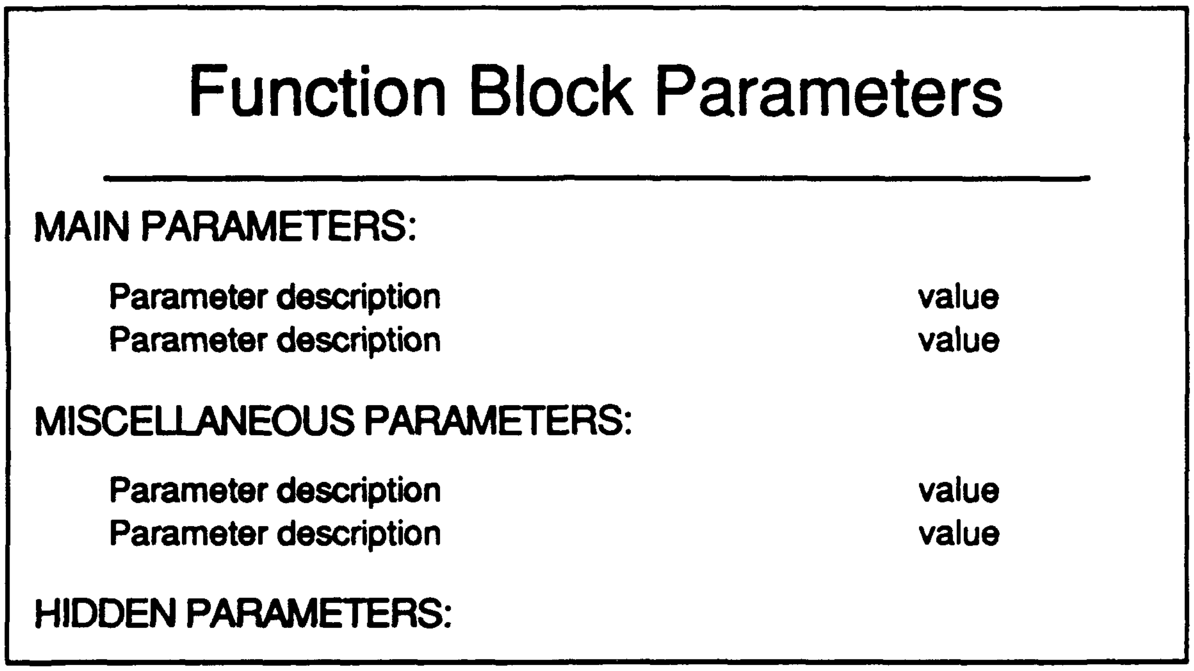


Figure 3-2 Function Block Symbol Model

3.1.2.2 <u>Parameter Model.</u>

The parameter model for a function block defines the state parameters that are different for each instance of the function block. The parameters can be used to define the initial configuration of the function block (e.g., number of layers and processing elements; learn rate, etc.); state parameters that will vary during the simulation (e.g., accumulated RMS error; number of passes; weights; etc.); miscellaneous parameters that define detailed options or variants of the function block (e.g., momentum; filter gains; scale factors, etc.) and hidden parameters which are not shown to the user but are used internal to the function block (e.g., neural network object; network configuration file; etc.).

The parameter model is displayed to the user in a separate window or screen that the user can edit or enter parameter values. Figure 3-3 illustrates a parameter screen, which is divided into sections. Each parameter has a name, and description that describes the entry and units for the parameter. Each parameter has a default value which is used if the user does not enter a new parameter value.



Function Block Parameters

MAIN PARAMETERS:

Parameter description	value
Parameter description	value

MISCELLANEOUS PARAMETERS:

Parameter description	value
Parameter description	value

HIDDEN PARAMETERS:

Figure 3-3 Function Block Parameter Model

For custom coded function blocks, the parameter model is displayed in a separate window linked to the symbol. The user entries to parameters in this window are passed to the function block during initialization to configure the specific instance of the function block. Also any changes to the state can be inserted back into the BDE database for latter viewing by the user using the BDE.

For custom user function blocks, the parameter model is displayed is a parameter screen associated with the detail model. The user entries to parameters in this window are passed on to the custom coded function blocks that make up the detailed design. This allows the user to edit or configure the low level function blocks without enter each's parameter model. Also it can be used to identify and propagate common parameter values to all constituents.

3.1.2.3 Detailed Model.

The detailed model represents the implementation for custom user function blocks, showing the detailed block diagram of the constituent function blocks associated with the symbol model. The detailed model identifies the constituent function blocks, their internal connections, and processing paths (see Figure 3-4). The detailed model is linked to the symbol model and can be accessed by the user by double clicking the symbol in the system block diagram. The user is able to convert any block diagram into a hierarchical function block with an associated symbol model using BDE.

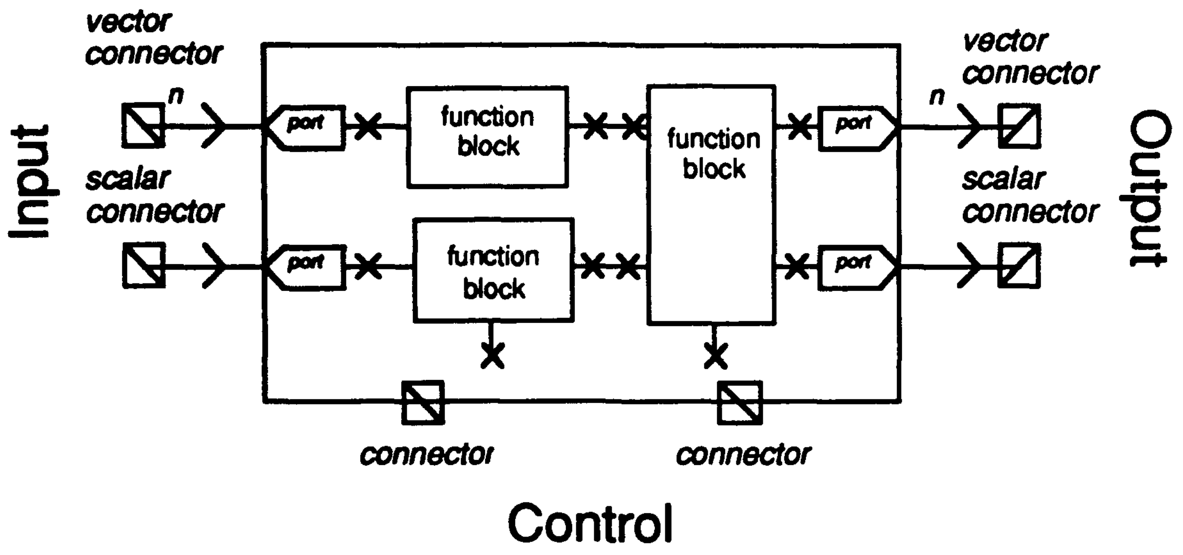


Figure 3-4 Function Block Detail Model

3.1.2.4 Block Model.

For custom coded function blocks, the block model represents the internal processing of the block during simulation. The template for the function block source code is generated from the symbol and parameter models. This establishes the relationship and interface to these models in the function block implementation. The template is then edited to include the custom code that will be called during the simulation. See paragraph 3.1.3 for a discussion of the allocation of processing resources to a function block during simulation.

3.1.2.5 Expression Model.

The expression model is the same as the block model except that expression model describes the code to be generated by the Code Generation System (CGS). The code generation can be customized to generate only the necessary code of a particular configuration of the function block based upon the instance's configuration parameters. Also the code generated may be optimized for a specific host platform or parallel configuration.

3.1.3 Memory And Processing Time Allocation

The memory and processing time for each function block instance is managed by the signal flow simulator during runtime. The Simulation Program Builder will reduce a hierarchical block diagram into a network list of low level function blocks removing all hierarchical (custom user) function blocks. The custom code function blocks are called during runtime to perform specific process tasks for an instance of the function block. The simulation will manage the memory allocation for all parameters and external signals to the function block. The function block will be given an opportunity to allocate, access, and dispose any internal parameters or objects for an instance. Figure 3-5 gives the SPB interface which shows the support from the NNOM in the implementation of a neural network function block.

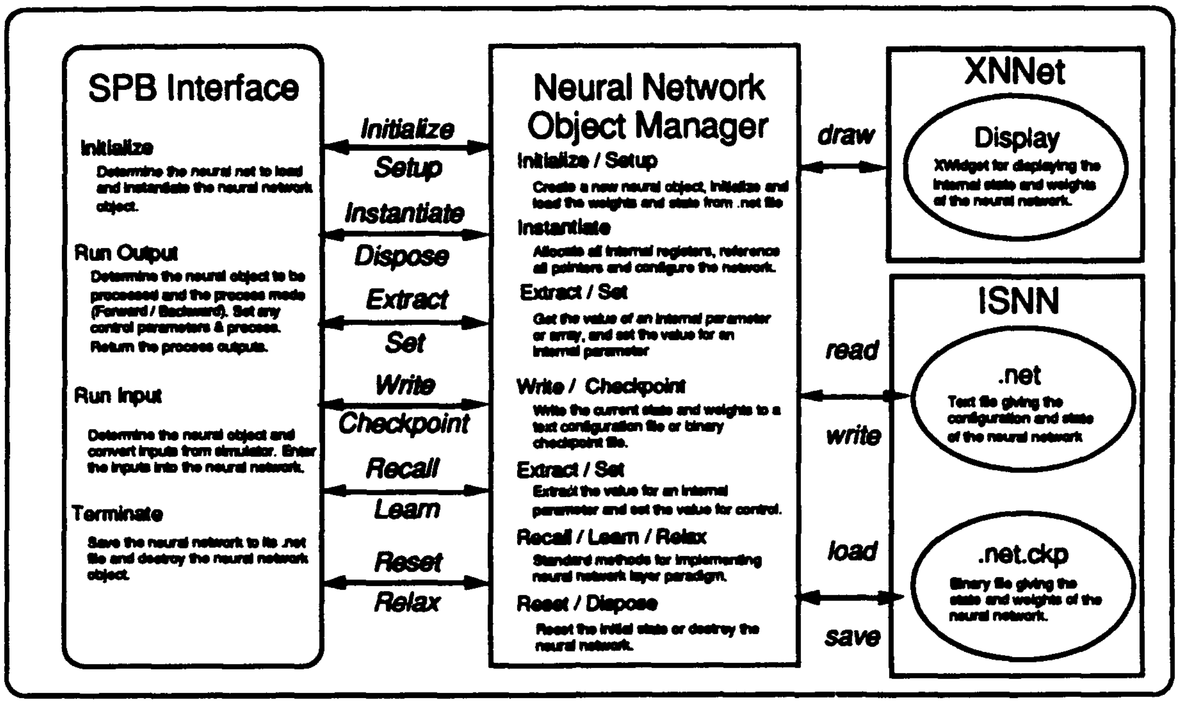

SPB Interface
Initialize
Determine the neural net to load and instantiate the neural network object.
Run Output
Determine the neural object to be processed and the process mode (Forward / Backward). Set any control parameters & process. Return the process outputs.
Run Input
Determine the neural object and convert inputs from simulator. Enter the inputs into the neural network.
Terminate
Save the neural network to its .net file and destroy the neural network object.
Initialize
Setup
Instantiate
Dispose
Extract
Set
Write
Checkpoint
Recall
Learn
Reset
Relax
Neural Network Object Manager
Initialize / Setup
Create a new neural object, initialize and load the weights and state from .net file
Instantiate
Allocate all internal registers, reference all pointers and configure the network.
Extract / Set
Get the value of an internal parameter or array, and set the value for an internal parameter
Write / Checkpoint
Write the current state and weights to a text configuration file or binary checkpoint file.
Extract / Set
Extract the value for an internal parameter and set the value for control.
Recall / Learn / Relax
Standard methods for implementing neural network layer paradigm.
Reset / Dispose
Reset the initial state or destroy the neural network.
draw
read
write
load
save
XNNet
Display
XWidget for displaying the internal state and weights of the neural network.
ISNN
.net
Text file giving the configuration and state of the neural network
.net.ckp
Binary file giving the state and weights of the neural network.

Figure 3-5 SPB/NNOM Interface.for NNCL Function Block

3.2 THE COMPUTER SOFTWARE COMPONENTS OF NNCL

3.2.1 Neural Network Object Manager (CSC01)

CSC Purpose. This CSC provides the standard interface between the SPW function blocks and the Industrial Strength Neural Network (ISNN) Library. Each neural network function block must have a representation in the BDE database and a corresponnding neural network object in the NNOM database. Figure 3-6 illustrates the interfaces for the construction of neural networks within the NNCSS.

Execution and control data flow. The neural network object management routines are implemented as standard C functions that are called from within custom SPW function block implementations. The primary object managed by the NNOM routines is a neural network object that can encapsulate other neural network objects as well as internal parameters and registers. The NNO's are created and setup by the NNOM from parameters defined in the BDE prior to a simulation run. The NNOM saves the NNO's to binary and text files for loading in subsequent simulations. Also the NNOM routines translate and report the ISNN error code status to SPW error reporting and processing.

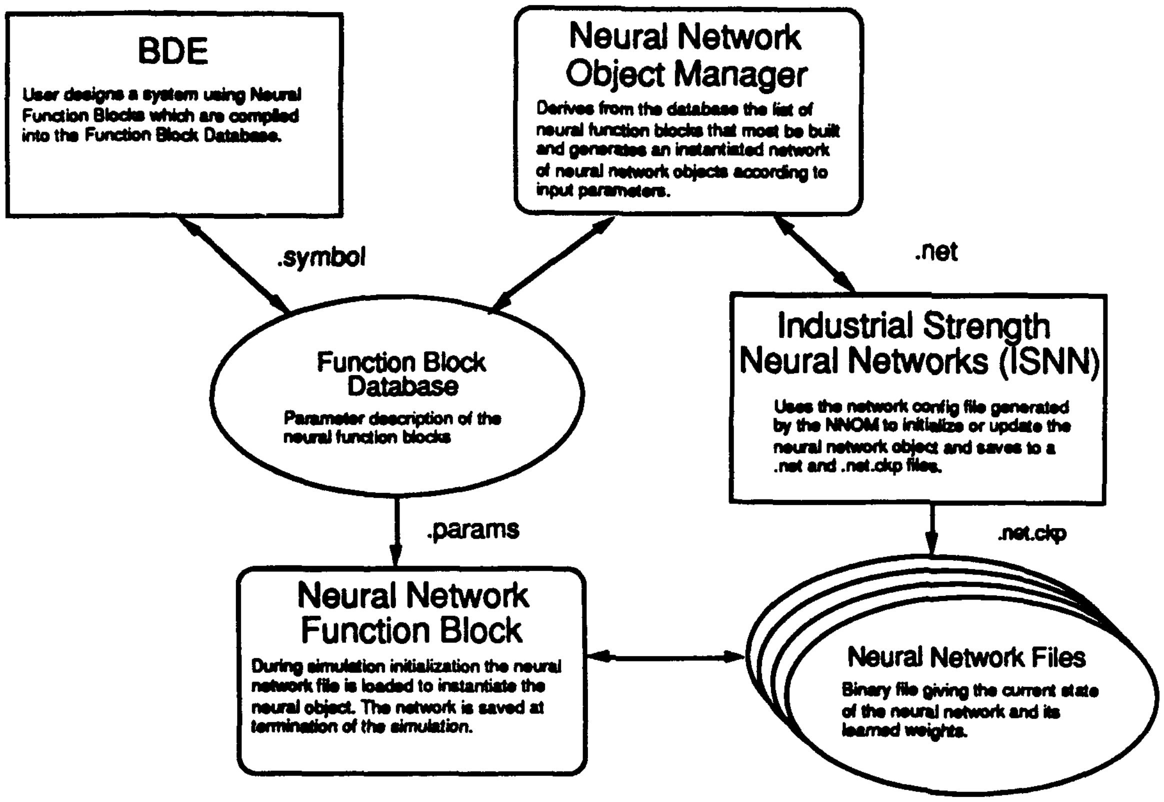

BDE
User designs a system using Neural Function Blocks which are compiled into the Function Block Database.
Neural Network Object Manager
Derives from the database the list of neural function blocks that most be built and generates an instantiated network of neural network objects according to input parameters.
.symbol
.net
Function Block Database
Parameter description of the neural function blocks
Industrial Strength Neural Networks (ISNN)
Uses the network config file generated by the NNOM to initialize or update the neural network object and saves to a .net and .net.ckp files.
.params
.net.ckp
Neural Network Function Block
During simulation initialization the neural network file is loaded to instantiate the neural object. The network is saved at termination of the simulation.
Neural Network Files
Binary file giving the current state of the neural network and its learned weights.

Figure 3-6 Neural Network Builder Interfaces

3.2.2 Neural Network Management (CSC02)

CSC Purpose. This CSC provides general function blocks to manage the processing of neural networks during a simulation. This includes custom coded function blocks for cycling the system through multiple passes of the input signal files for embedded neural network training; and general custom user function blocks to define typical configurations of neural network and preprocessing function blocks.

Execution control and data flow. Figure 3-7 gives the general function block architecture for management of neural network objects within the NNCSS. The NNOM function block creates an object manager to contain the neural network objects. At the top level the NNOM manages all neural network objects including instantiation, checkpointing, writing, and disposal of the system of neural network objects. At system initialization, it either loads a previously saved neural network configuration or creates a nnom to manage the neural network objects created within the simulation. The resulting nnom is output to all neural network objects.

The neural network object function block is typically a specific neural network paradigm process. The neural network object function block either retrieves a previously saved neural network object from the nnom or creates a new neural network object to be managed by the nnom. The function block uses the BDE defined parameters to setup the function block during system initialization. After all the function blocks have setup their neural network objects, the NNOM function block allocates and instantiates all of the objects at the start of processing. During the simulation, the NNOM periodically saves binary versions of the neural network objects during checkpointing. At the termination of the simulation, the top level NNOM writes the system to a text configuration file.

Each neural network object has control inputs that define the processing state of the object. These include hold - suspends processing whenever hold is greater than 0.0; learn - activates the paradigm learning algorithm when greater than 0, and relax - performs the learn epoch update processing during the pass.

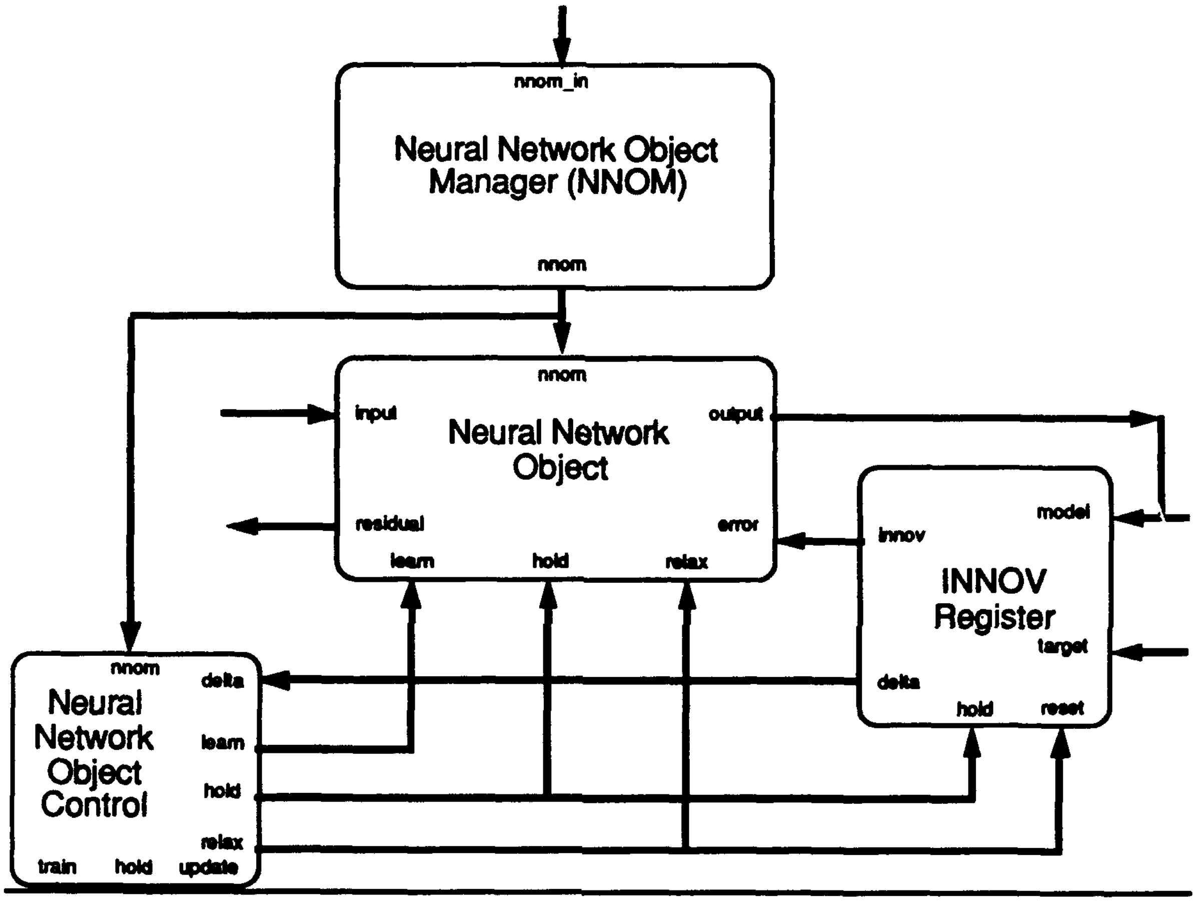


Figure 3-7 - Neural Network Object Management Component Architecture

The Neural Network Object Controller (NNOC) function block coordinates these control signals during a simulation. It generates the appropriate learn and relax control signals to implement multiple training epochs during a simulation. It supports feedback of a delta error signal, such that the rms or max output error from the innov register can be used to threshold the learning during the backward pass through the neural network object. The NNOC is used primarily for controlling supervised neural network objects.

For unsupervised learning, the Neural Network Object Control Clock (NNOC_CLOCK) (not shown) is used to generate an inner and outer loop counter signal that allows various neural network objects to be processed at a higher rate than the other function blocks. This allows the internal cycles of the neural network object to be observed. The outputs from the NNOC_CLOCK are control signals that are connected to the function block's hold pin. The outer loop (pass) control signal is held;

the function blocks on the inner loop (cycle) are activated for a simulation cycle. At each epoch, the pass control signal is held down so that the other function blocks can execute.

3.2.3 Register Function Blocks (CSC03)

CSC Purpose. This CSC extends the standard library of vector function blocks to provide activation function processing commonly required for neural network processing. This includes normalization, shift, delay, window, merge, split, and error registers used in neural network designs.

Execution and control data flow. The activation function blocks are special vector processing functions, which are used to prepare the activation data for a neural network paradigm. These are based upon existing vector operators in the DSP Communications Library. There will be additional vector operations required to connect neural network function blocks within a communications system. These include:

a. *Shift Register*. A vector input is shifted onto the top of an array such that the last, t, inputs are represented within the register. The full register is output at each time step, providing a fixed window sample of the time sequence of input vectors.

b. *Delay Register*. The same as a shift register except that the bottom vector sample in the register is output with each time step, effectively delaying the output by the number of shifts in the register.

c. *Merger Register*. Combines vector inputs of different sizes into a composite vector at the output.

d. *Split Register*. Splits a larger vector into two smaller vectors.

e. *Offset Register*. Accesses a vector at a constant offset from a larger vector.

f. *Normalization and Denormalization Registers*. Perform the required normalization and denormalization in and out of a neural network function block.

3.2.4 Neural Network Instruments and Probes (CSC04)

CSC Purpose. This CSC extends the ISL to provide custom user function blocks that are particularly useful for the display and control of neural network simulations. These blocks provide instruments and probes to assist in the training and validation of neural network based simulations.

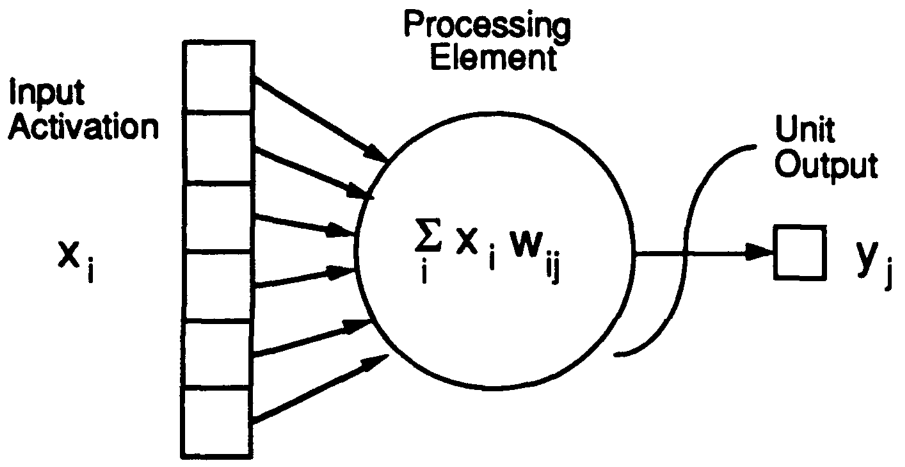
Execution and control data flow. These function blocks shall be implemented using ISL low level function blocks and will follow the execution and control data flow defined for these latter functions. Function blocks which will be controlled by these instruments will provide the appropriate control pins for attaching the instruments with a system design. General data extraction and data set function blocks shall be used to extract or control internal parameters and vectors within a neural network object.

3.2.5 Backpropagation Function Blocks (CSC05)

CSC Purpose. This CSC provides custom function blocks for implementing the Backpropagation Neural Network within a communications design. The CSC includes custom coded function blocks that implement forward and backward processing algorithms, and multilayered feed forward neural networks. It also provides custom user function blocks which allow backpropagation to be incorporated in various configurations and to perform various functions within a communications design.

Execution and control data flow. The execution and control data flow is defined by the propagation architecture of the Backpropagation Neural Network. The following defines the overall architecture to be implemented by this CSC.

The Backpropagation Neural Network is one of the most important and widely applied ANS paradigms. The backpropagation network is a form of *supervised* learning. Figure 3-8 illustrates the network architecture. The network is formed of one or more layers of processing elements. The first layer represents the input pattern and the output of these elements are connected to the input of each of processing elements in the next layer and so on through the network. As the signals feed forward through the network,



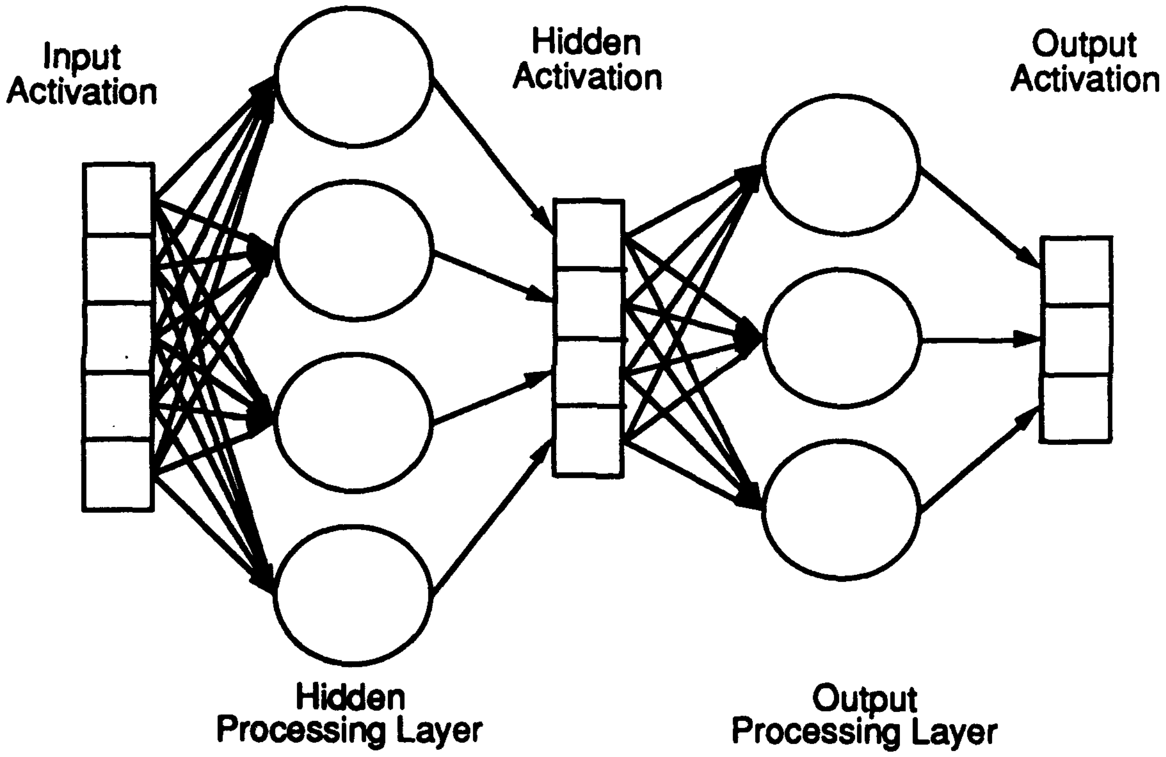
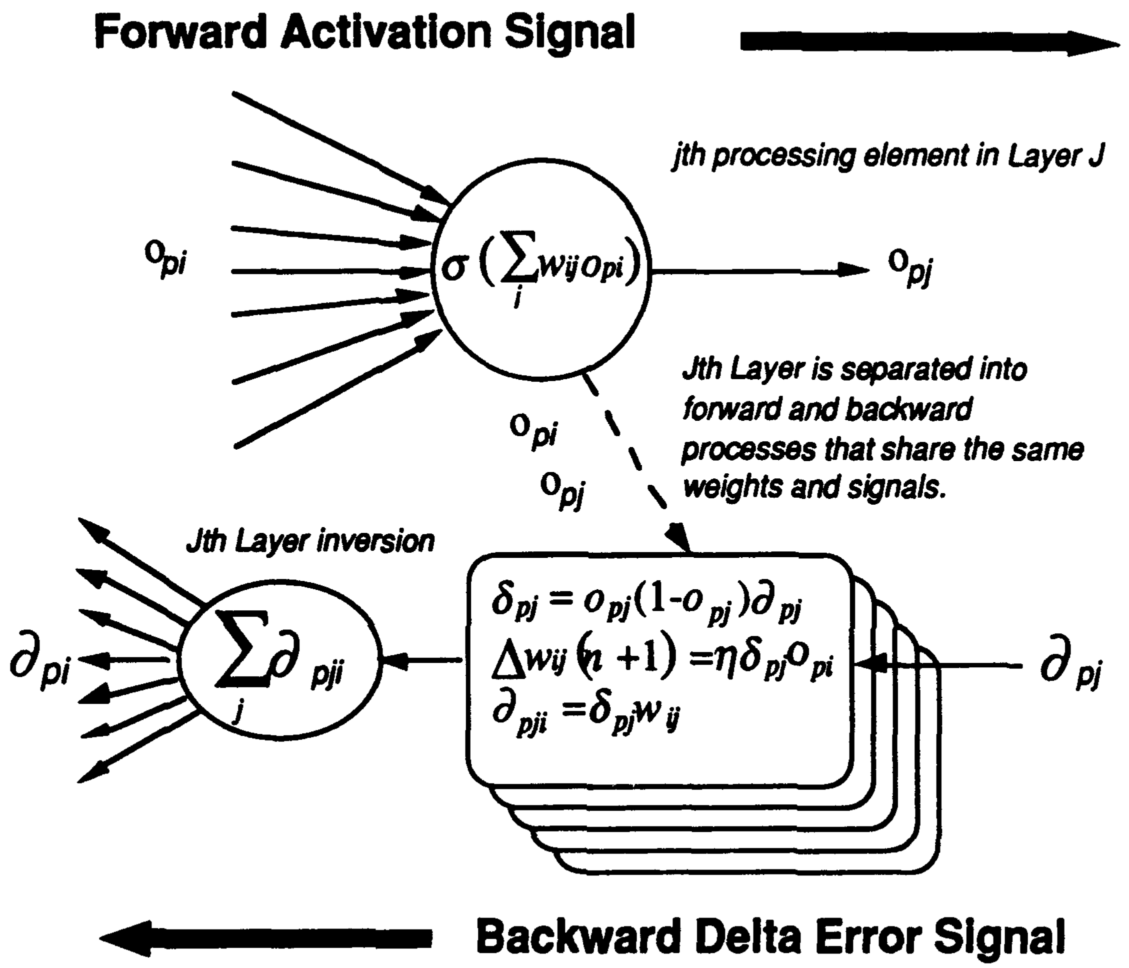


Figure 3-8 Backpropagation Network Architecture

each processing element takes the weighted sum of the activation input signals and passes this sum through a sigmoid threshold function to generate one output signal to each of the elements in the next layer. The output from the last layer in the network (called the output layer) represents the learned or retrieved pattern.

Initially the connection weights are set to random values. During training, example input patterns are propagated through the network. The output pattern is compared to the desired target pattern. The difference between the output and target patterns is the error which is back-propagated through the same connections. During backpropagation each processing element adjusts its connection weights slightly and propagates a delta error to the preceding layer of processing elements. The next time the input examples are propagated through the network, the output pattern is generally closer to the target pattern. The error is again back-propagated and the training cycle repeated until the total RMS error over the training set converges to an arbitrarily small value. The backpropagation network implements *gradient descent* learning which means that the weights are adjusted such that the total RMS error decreases to zero.

Figure 3-9 gives the learning algorithm for the network. It illustrates the algorithm for a processing element as two processing paths --the path of the forward activation signal



Forward Activation Signal
jth processing element in Layer J
o_{pi}
$\sigma(\sum_i w_{ij} o_{pi})$
o_{pj}
Jth Layer is separated into forward and backward processes that share the same weights and signals.
o_{pi}
o_{pj}
Jth Layer inversion
∂_{pi}
$\sum_j \partial_{pji}$
$\delta_{pj} = o_{pj}(1-o_{pj})\partial_{pj}$
$\Delta w_{ij}(n+1) = \eta \delta_{pj} o_{pi}$
$\partial_{pji} = \delta_{pj} w_{ij}$
∂_{pj}
Backward Delta Error Signal

Figure 3-9 Backpropagation Learn Algorithm for a Layer

and the path of thebackward propagated delta error signal. The two processing paths share the same connection weights between weight changes. The input, output and target patterns of the forward signal path are used to compute the changes to the weights during the backward processing. This algorithm is performed on all of the processing elements (j) within a layer.

The delta error backpropagated from a single processing element is a vector that corresponds one-to-one with the input activation vector. In turn, each processing element in a hidden layer receives a delta error component from each processing element in the next layer to which it is connected. During backpropagation every weight in the entire network receives a specific delta error which is used in the equation to change that weight.

For optimal gradient descent learning, the delta weight changes are accumulated for a designated number of cycles, and then applied to change the connection weights at the end of each update interval. To train the neural network requires multiple passes through the training data or multiple training epochs through the input stream.

3.2.6 Kohonen Feature Map Function Blocks (CSC06)

CSC Purpose. This CSC provides custom function blocks for implementing and training Kohonen Topological Feature Map neural networks. The CSC includes custom coded function blocks for unsupervised learning and recall processing algorithms. It also provides custom user function blocks that allow Kohonen networks to perform various functions within a communications design.

Execution and control data flow. The execution and control data flow is defined by the propagation architecture of the Kohonen Topological Feature Map. The following defines the overall architecture to be implemented by this CSC.

The Kohonen Topological Feature Map provides a method for creating networks that can be trained to classify input vectors while preserving the inherent topology of the training set. Topological preserving maps mean that the nearest neighbor relationships in the training set are preserved in the network such that input vectors presented to the network that have not been previously "learned" will be categorized by its nearest neighbor in the network's learned exemplar set.

Such a network becomes a parameterized version of a Bayesian classifier, except that the probability distribution does not have to be known a priori. The only requirement is that the training set has to be a representative sample of the distribution of input patterns. Once a network has been trained, it will provide near optimal performance in classifying input patterns with minimal processing.

Its ability to map unknown distributions can extend the use of the network to areas where Bayesian approaches are intractable - such as voice recognition and vision processing. It can be used for feature extraction within a more complex network such as counter propagation, or as a filter in a signal processing network.

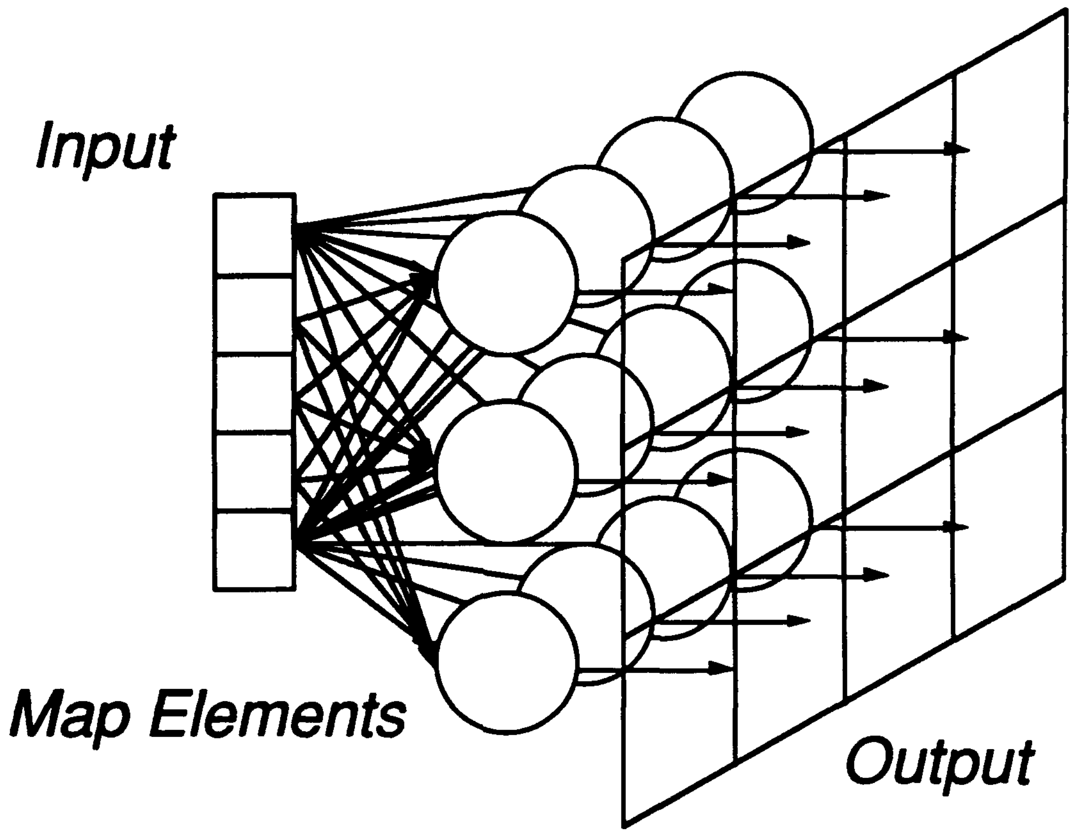


Figure 3-10 Kohonen Topological Feature Map Architecture

Figure 3-10 gives the network architecture. The network consists of an input vector of dimension N that is fully connected to a set of M processing elements, where M is the number of exemplar vectors for the mapping. The M processing elements are arranged to form a two dimensional array at the output. The dimensions of the array can be any size, but are typically arranged to form a square array with $M^{1/2}$ elements per side. The output from the network is a vector of M elements given by:

$$y_k = \frac{1}{2}|w_k - x|^2 - \frac{1}{2}|x|^2 \qquad k = 1,\dots, M \qquad 3.2.6\text{-}1$$

where the winning node y_j is the minimum element value:

$$y_j = \min \{y_1, y_2, \dots, y_M\} \qquad 3.2.6\text{-}2$$

giving the nearest exemplar vector (w_j) to the input vector (x). The network can also work when maximum activations are used during training. In that case, the exemplar vectors are in the hyperplane normal to the input vector.

During training the exemplar vectors are arranged into a two dimensional topological map where the output values will increase proportionally to the distance from the minimum. This two dimensional configuration gives the maximum degree of freedom for representing nearest neighbor relationships that are not strictly ordered. A strictly ordered relationship requires that the distance among the exemplar categories be transitive (A < B < C implies A < C and that B is between A and C). With square arrays, weakly ordered and quasi-ordered relations can be accommodated. The ordering can have a geometric interpretation for simple two-dimensional vectors. For example, if the input vector has two dimensions whose element values are distributed uniformly over the interval [0.0 to 1.0], then the set of exemplars will span the space uniformly such that the output values increase proportionally to the distance from the minimum element in the output array. For higher dimensional input vector spaces, such a geometric interpretation is not possible. The resulting output array will be a minimum spanning tree where regions of the array will become excited by the input vector according to their distance relative to this relational metric.

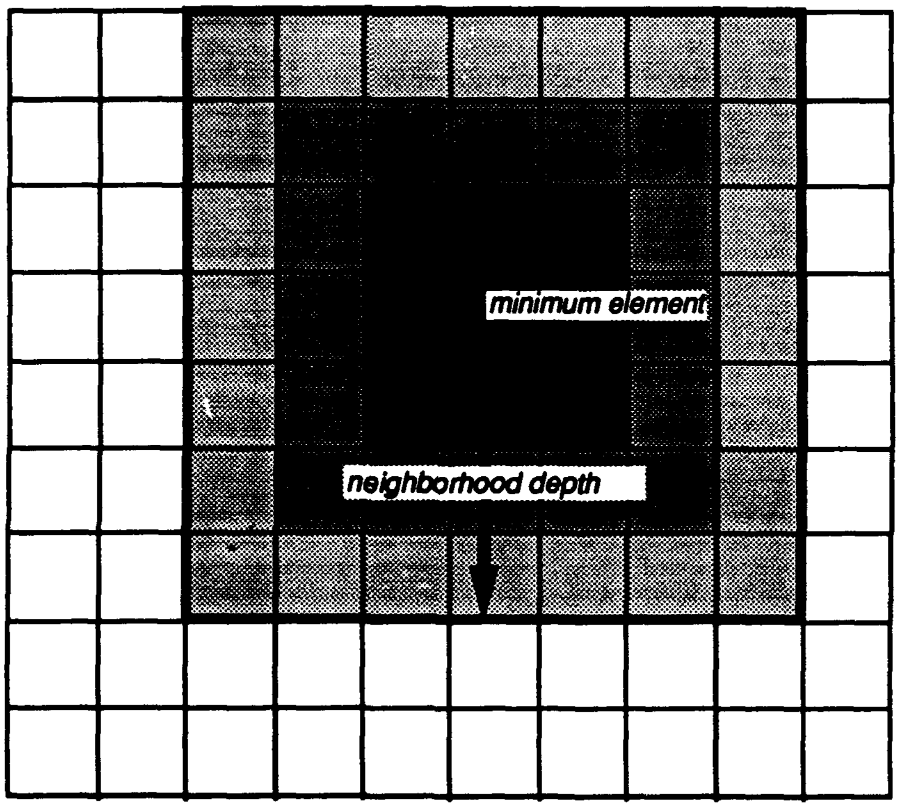


Figure 3-11 Kohonen Neighborhood Diagram

To achieve a parameterized topological map, the network is presented with a training set, which is a random sample of the values that the elements of the input vector can have. Initially, the exemplar weights are randomized unit vectors. When an exemplar vector is found nearest to the input training vector, the other vector weights are updated in proportion to their distance from the minimum in the output array for a neighborhood about the minimum. The dynamics of the weight equation are given by:

$$\Delta w_{ij} = \alpha \left(x_i - w_{ij}\right) \qquad 3.2.6\text{-}3$$

where j is in a neighborhood of the jth node in y, and a is the learn rate for the weight connections. The neighborhood is defined by a neighborhood depth, giving the number of nodes from the minimum to be updated. This is illustrated in Figure 3-11. At first, the entire array of exemplar vectors is adjusted about the minimum vector for each input. After a number of iterations, the region about the minimum is gradually reduced until only the nearest neighbors are updated. Subsequent iterations refine the exemplar vectors by reducing the learning rate according to:

$$\alpha = \alpha_\rho \left(1 - \frac{t_\rho}{T_\rho} \right) \qquad 3.2.6\text{-}4$$

where ρ designates the learning phase, T_ρ is the maximum number of interations within the current learning phase, t_ρ is the number of iterations since the beginning of the learning phase, and α_ρ is the initial learn rate for the phase.

At the end of a learning phase, $t_\rho = T_\rho$, the maximum time is increased by a factor of 5, Tp = 5Tp-1, and the learn rate is decreased by a factor of 5, $\alpha_\rho = \alpha_{\rho-1} /5$. Likewise, during the initial phase, $\rho = 0$, the neighborhood depth, ω, is decreased as a function of the interations by:

$$\omega = \omega_o \left(1 - \frac{t_o}{T_o} \right) \qquad 3.2.6\text{-}5$$

where ω_O is the initial neighborhood depth, set to one-half of the output array side dimension. In subsequent phases, $\omega = 1$, indicating only the nearest neighbors are updated.

Typically, 100,000 iterations are required to achieve an optimal mapping function, however the basic topology is achieved during the initial training phase.The resulting parameterized map will be a set of exemplar vectors of near-uniform length that will model the input probability distribution. The exemplars will be arranged such that each exemplar is equally likely to capture an input vector. Regions of high probability will have more exemplar vectors than regions of low probability.

3.2.7 Fully Recurrent Network Function Blocks (CSC07)

CSC Purpose. This CSC provides custom function blocks for implementing and training fully recurrent back propagation neural networks within an embedded system design. The CSC includes custom coded function blocks that implement forward recall and error feedback learning algorithms. It also provides custom user function blocks which allow recurrent networks to be incorporated in various configurations and to perform various functions within a communications design.

Execution and control data flow. The execution and control data flow is defined by the propagation architecture of the Fully Recurrent Network. The following defines the overall architecture to be implemented by this CSC.

The Fully Recurrent Network is similar to the Backpropagation Network except that the neural network learns to map sequences at the input into sequences at the output. A fully recurrent network consists of a set of processing elements which are fully connected with every other processing element in the network and with the input activation vector. Figure 3-12 illustrates the fully recurrent network architecture as developed by William and Zipser.

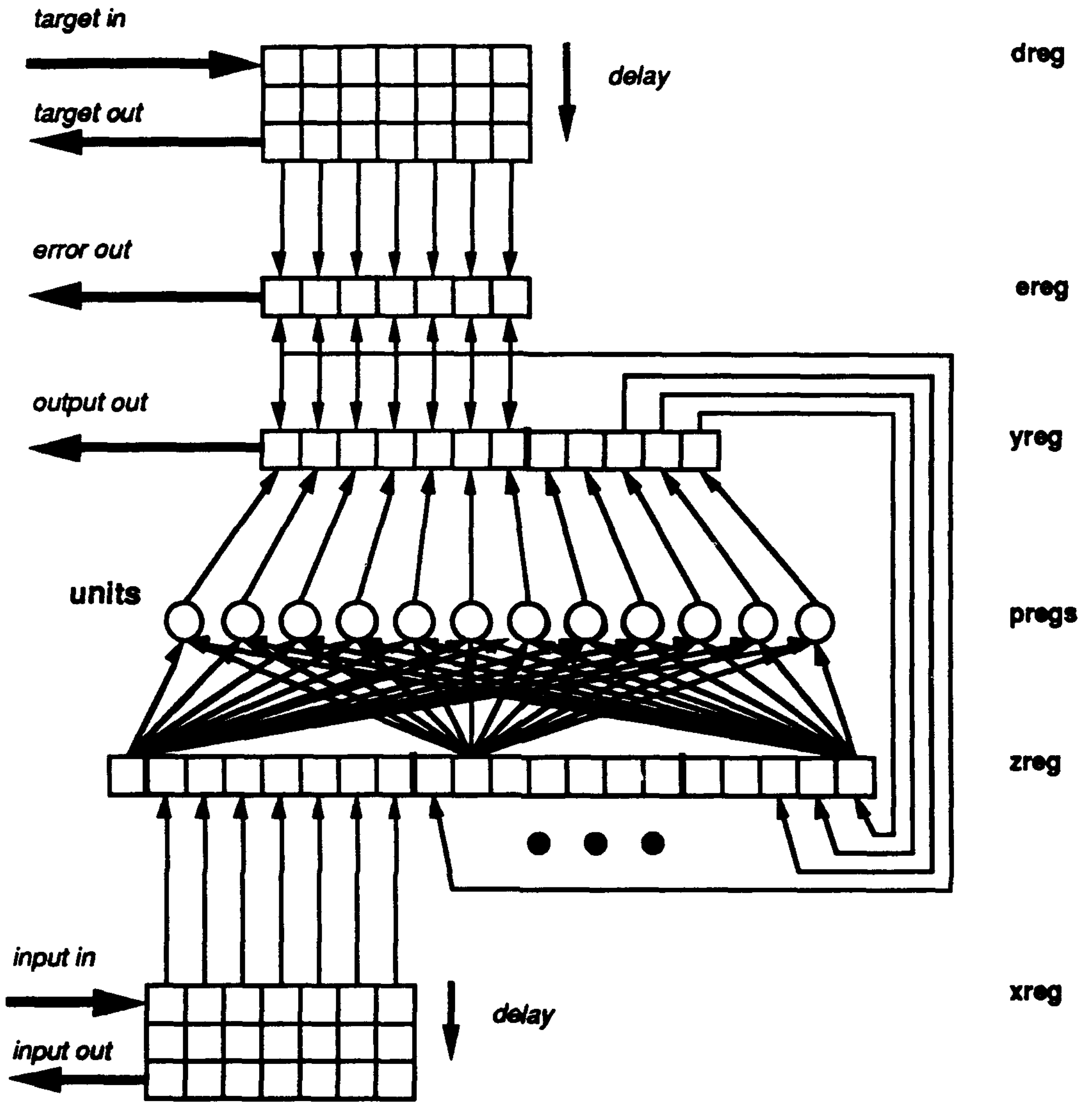


Figure 3-11 Fully Recurrent Network

The input to the network, $z_k(t)$, consists of the input activation vector, $x_k(t)$, combined with the output activation vector of the previous cycle from all of the processing elements, $y_k(t)$:

$$z_k(t) = \begin{cases} x_k(t) & \text{if } k \in I \\ y_k(t) & \text{if } k \in U \end{cases} \qquad 3.2.7\text{-}1$$

where $k = 1 \ldots M + N$ where M is the number of external inputs and N is the number of processing units. The sets I and U designate the input and output indices respectively.

The output for the kth element at the next time step, $y_k(t+1)$, is given by the sum of the weighted connections for the kth processing element, $s_k(t)$, passed through the unit's squashing function f_k:

$$s_k(t) = \sum_{l \in U} w_{kl} y_l(t) + \sum_{l \in I} w_{kl} x_l(t) = \sum_{l \in U \cup I} w_{kl} z_l(t) \qquad 3.2.7\text{-}2$$

$$y_k(t+1) = f_k(s_k(t)) \qquad 3.2.7\text{-}3$$

The recurrent application of the output to the input allows the network to form its own internal organization for learning time sequences at the input (i.e. recurrence of patterns at the input) and it also allows the network to form its own internal layers over several iterations of the input pattern.

The output from the network consists of a subset of the processing element outputs (yreg) which are trained from target vectors (dreg). The processing elements which are not directly trained are used internally to the network as hidden activations. Let *T(t)* designate the set of active indices for the output, $y_k(t)$, at time t for which there is a corresponding target element, $d_k(t)$, then the delta error, $e_k(t)$, is:

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{if } k \in T(t) \\ 0 & \text{otherwise} \end{cases} \qquad 3.2.7\text{-}4$$

The delta error is back propagated to all of the units and the weights updated using:

$$\Delta w_{ij}(t) = \alpha \sum_{k \in T(t)} e_k(t) p^k_{ij}(t) \qquad 3.2.7\text{-}5$$

where α is the learning rate and $p^k_{ij}(t)$ is kth unit's contribution to the weight change given by:

$$p^k_{ij}(t+1) = \frac{\partial y_k(t+1)}{\partial w_{ij}} = f'(s_k(t))\left[\sum_l w_{kl} p^l_{ij} + \delta_{ik} z_j(t)\right] \qquad 3.2.7\text{-}6$$

Since p^k_{ij} incorporates the weighted sum of all other p^l_{ij} then equation 3.2.5-5 distributes the error, $e_k(t)$ over all of the weights, w_{ij}, including the input connected weights. Each processing element accesses the entire connection weight matrix and accumulates delta weight adjustments in internal registers (pregs) for each weight and each processing element . At the end of each input time step the weights are adjusted in preparation for the next input. Figure 3-13 shows the process block diagram for a training cycle.

Because the network is conditioned by both the input activations and previous output activations, there are a number of options which can be employed during training. The first has already been mentioned and involves the update of the weights at each input iteration. This allows the network to train in real-time on a stream of input activations without accumulating delta weight changes. As a result the network can be applied to streams of various lengths in real time without having to define epochs often decreasing by orders of magnitude the number cycles required to train the network. This is referred to as the realtime option.

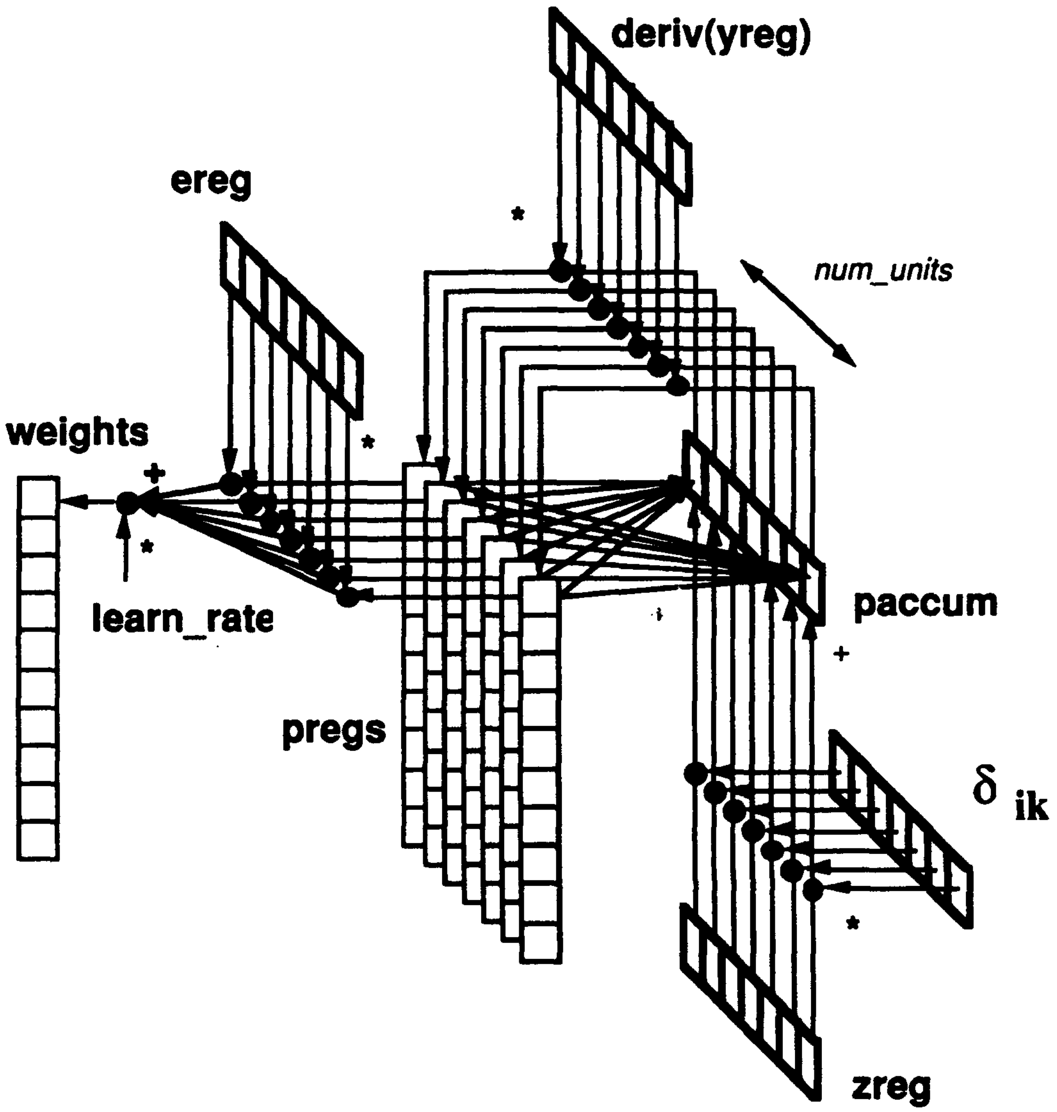


Figure 3-12 Fully Recurrent Network Dynamics for Learning Algorithm

The shifting of the output to the input for the next input iteration presents two options. The first is to recycle the output generated by the network, and the second is to replace the generated output with the target output for the next cycle. The latter is referred to as forced learning. With forced learning, the network can learn sequences which would be impossible using the generated outputs. These situations involve learning to bifurcate an input value to more than one output value depending upon previous history of output activations. For example, learning a sine wave, the network can distinguish between the value 0.74 predicting a higher or lower value depending upon the previous output being lower or higher. Without forced learning the network will attempt to converge on a value which does minimizes the error between the two predicted values such as 0.74.

Another important option is that the target vector can change over the processing elements so that different subsets of processing elements are selected as output activations at each time step. This allows certain processing elements to be specialized to certain target events while still being available internally as hidden processing elements for other events. Because the processing element is sometimes an output processing element and other times a hidden element, its behavior is different from those that are strictly output or hidden. These processing elements would attempt to form weight connections which attract certain patterns representing the target and repel at various degrees other patterns which do not map to the target. This implies that these processing elements can be self organizing similar to an adaptive network such as a Kohonen topological feature map.

Another useful option is to delay the application of a target vector for a specified number of cycles for an input. In Figure 3-12, this is shown as shift registers for the input and target activations. The input and target vectors are placed at the top of the shift register and shifted down with each new input/target pair. The input vector is taken from the top of the input shift register and the generated output is compared to the target at the bottom of the target shift register. The output corresponds to the input vector placed on the shift register n cycles previously. The network is process and trained just as before except that the output is delayed. What this option does is cause the hidden processing elements to form layers similar to backpropagation layers.

3.2.8 Adaptive Resonance Theory Function Blocks (CSC08)

CSC Purpose. This CSC provides custom function blocks for implementing and training Adaptive Resonance Theory (ART 1, 2 and 3) neural networks within embedded system designs. The CSC includes custom coded function blocks for unsupervised learning and recall processing algorithms. It also provides custom user function blocks which allow ART networks to perform various functions within a communications design.

Execution and control data flow. The execution and control data flow is defined by the propagation architecture of Adaptive Resonance Theory. The following defines the overall architectures to be implemented by this CSC.

Adaptive Resonance Theory - Binary (ART 1)

ART 1 is the first paradigm developed by Carpenter and Grossberg based upon the adaptive resonance theory by Grossberg. In the latter paper, Grossberg investigated the instability of non-linear systems that incorporate feedback, and determined stable processes that are adaptive and self-organizing. ART 1 is applied to binary input vectors (i.e., the elements of the vector are either 0.0 or 1.0).

ART 1 Architecture. The network consists of two sets of nodes. The first set, F1, is connected to the input vector and presents a short term memory (STM) activation vector at its output. The second set, F2, generates an output activation vector giving the recalled category from long term memory (LTM) for the current input vector. Initially, the STM from F1 is activated by the input vector. This pattern activates all of the nodes in LTM concurrently via bottom-up weights from F1 to F2. These activations compete until F2 becomes active with a candidate category of a pattern stored in LTM. The F1 STM then receives the recalled LTM pattern via top-down weights from F2 to F1. If this pattern is not sufficiently close to the input pattern, a strong inhibit signal is sent to F2. This suppresses the winning category and another category becomes active. If the new category matches the input pattern, the system becomes stable and resonates for a sufficient time interval, such that the LTM pattern is reinforced by the new input pattern and learning occurs.

The learning process involves updating the bottom-up and top-down weights representing the LTM patterns. If an input pattern has not previously been presented to the network and is sufficiently different for the other input patterns stored in LTM, the new pattern is eventually introduced to LTM. This ability to learn new patterns while distinguishing previously learned patterns represents the adaptive feature of ART1. ART1 is also self-organizing. The categories are selected and the members of the those categories are determined by the properties of the network and not from an external target vector as is the case of supervised learning paradigms such as back propagation.

The properties of an ART1 network are defined by a set of parameters representing the coefficients of the general dynamic equations for the network. These dynamic equations are based upon a dimensionless form of the membrane equations developed by Lin and Segal. Though the number of parameters has been reduced to a minimum, there remains a relatively large number that must be defined to achieve a stable network. Therefore, tuning an ART1 network to a particular problem requires understanding the dynamics of the network from the equations which define the network. The essential equations are presented without derivation to allow the user to experiment with the model and develop intuition as to the model's performance under various parameter settings. For a more complete theoretical understanding of the properties of the ART1 network and as a foundation to the other ART networks refer to the referenced papers in Chapter 2.

ART 1 STM Equations. We denote the activation vectors from F1 and F2 with the subscripts i = 1, 2, 3,...,N and j = 1,2,3,...,M respectively. N is the number of elements in the input vector and M is the total number of available categories in LTM. The parameters A, B, C,... for layers F1 and F2 are denoted with subscripts 1 and 2, respectively. The equations governing the dynamics of the activation vectors $x_i(t)$ and $x_j(t)$ for F1 and F2 are:

$$\varepsilon\frac{d}{dt}x_i=-x_i+(1-A_1x_i)J_i^+-(B_1+C_1x_i)J_i^- \qquad 3.2.8\text{-}1$$

$$\varepsilon\frac{d}{dt}x_j=-x_j+(1-A_2x_j)J_j^+-(B_2+C_2x_j)J_j^- \qquad 3.2.8\text{-}2$$

where J_k^+ is the total excitory input at unit k = i or j, and J_k^- is the total inhibitory input to unit k. All of the parameters are non-negative. If A > 0 and C > 0, then the elements of the activation vectors, xi(t) and xj(t), will remain in the finite interval $[-BC^{-1}, A^{-1}]$ regardless of how large the non-negative inputs J_k^+ and J_k^- become.

The excitory input J_i^+ for the ith node of F1 is the sum of the bottom-up input I_i and the top-down template input from F2:

$$J_i^+=I_i+D_1\sum_j f\,(x_j)z_{ji} \qquad 3.2.8\text{-}3$$

where $f(x_j)$ is the threshold signal generated for an F2 node by the activation x_j, and z_{ji} is the top down LTM weight from F2 to F1.

The inhibitory input J_i^- is derived from all active nodes in F2:

$$J_i^- = \sum_j f(x_j) \qquad \text{3.2.8-4}$$

If F2 has at least one element active, then $J_i^- > 0$ and has a non-specific inhibitory effect on all of the units of F1.

The excitory input J_j^+ to the jth node of F2 comes form the bottom-up trace of activations from F1 and the positive feedback signal g(xj) to itself:

$$J_j^+ = g(x_j) + D_2\left(\sum_i h(x_i) z_{ij} - a_j^+\right) \qquad \text{3.2.8-5}$$

where $h(x_i)$ is the threshold signal generated by F1 at the activation xi and zij is the bottom-up LTM weights from F1 to F2. Here the weights z_{ij} and z_{ji} denote two different matrices and not the transposition of the same matrix. The vector element a_j^+ is the excitory activation from the orienting subsystem which indicates the category is a learned category. If biases the competition to first find categories which have been previously learned before starting a new category for the input vector. This is discussed further later.

The inhibitory input J_j^- to the jth node of F2 is the competive negative feedback from all of the other nodes in F2:

$$J_j^- = \sum_{k \neq j} g(x_k) - a_j^- \qquad \text{3.2.8-6}$$

where a_j^- is the inhibitory activation signal from the orienting subsystem when the category does not match the current input pattern. When active, it suppresses the winning category so that other categories will be considered. The dynamics of the reset or orienting subsystem are discussed in the subsection entitled "Orienting Subsystem Equations" for ART 1.

ART 1 LTM Equations. The dynamics for the LTM weights are as follows:

$$\frac{d}{dt} z_{ji} = f(x_j)\left[-z_{ji} + h(x_i)\right] \qquad \text{3.2.8-7}$$

$$\frac{d}{dt} z_{ji} = K_2 f(x_j)\left[(1 - z_{ij}) L_2 h(x_i) - z_{ij} \sum_{k \neq i} h(x_k)\right] \qquad \text{3.2.8-8}$$

When a stable resonant configuration in the STM presists, the weights are updated by the above dynamic equations. For a stable network, it is important that the weights are modified at a time scale that is long compared to the update of the STM activations. This is achieved by keeping the time scale for the learning, δt, small while allowing E to be large enough such that $E\delta t = 1$. After the input activation hasn been presented for a sufficient interval to indicate that the system is resonating, the ART1 model solves the dynamic equations and updates the weights to their asymptotic values for $t >> 0$. This is referred to as the "quick learn mode". The quick learn mode is engaged when the following condition is satisfied:

$$counter = {}^{2.5}\!/_{E_1\delta t} \qquad 3.2.8\text{-}9$$

where dt is the time constant and counter is the number of iterations since the last reset. E is a parameter of the ART 1 implementation that controls LTM learning rate and is not originally a Grossberg ART 1 parameter.

ART 1 Orienting Subsystem Equations. The orienting subsystem compares the STM recalled pattern to the input vector and becomes active when the comparison falls below a vigilance threshold. The vigilance is set within the range 0.0 to 1.0 where 1.0 indicates that the STM and input patterns must match exactly and lower values allow proportionally greater dissimilarity among the members with a category. Higher vigilance will result in greater discrimination among the input vectors, whereas lower vigilance will tend to group the input patterns with greater variation into the same category.

When a mismatch is found, the orienting subsystem is "aroused" and an inhibitory signal is transmitted to F2 which has the following dynamics:

$$\frac{d}{dt}a_j^- = G_2\left(-a_j^- + \alpha g(x_j)\right) \qquad 3.2.8\text{-}10$$

where α is the arousal level for the inhibitory signal and is a function of the vigilance. When the arousal is greater than zero, the active element kj transmits a self-inhibiting signal until it is zero, at which time the inhibitory signal decays at a time scale of G2*δt.

When the orienting subsystem is not aroused, an excitory activation signal is sent for all categories that have been learned by the network. The equation for this process is:

$$\frac{d}{dt}a_j^+ = G_1 g(x_j)\left(-a_j^+ + \left(\frac{K_2\rho}{L_2 - 1 + \sum_i h(x_i)}\right)\sum_i h(x_i)\right)(1-\delta(\alpha)) \qquad 3.2.8\text{-}11$$

where ρ is the vigilance, β is the average initial value for the bottom-up weights, which is leass than L2/(L2 - 1 - N), and δ(α) = 1 when α > 0, and δ(α) = 0 otherwise. This sends a self-excitory signal to the already learned categories slightly above the average signal, β, of the unlearned or initialized categories. Therefore, the competion is biased to the learned categories for initial searches, but still allows new categories to be generated from the uninitialized weights. The effect of choosing L2 large is to bias the network to choose uncommitted category elements in response to unfamilar input patterns.

Adaptive Resonance Theory - Gray Scale (ART 2)

ART 2 applies the adaptive resonance theory developed by Carpenter and Grossberg to input vectors whose elements can vary continuously over the range from 0.0 to 1.0. ART 2 is an example of unsupervised learning where the output represents the category for the input pattern determined by the network itself. No target vector is provided during training. ART2 is also an example of adaptive learning in that when new patterns are presented, which are significantly different form the previously learned patterns, the network will recognize that the patterns are different and form new categories.

ART 2 Architecture. The architecture for ART2 is shown in Figure 3-14. The sets of nodes, F1 and F2, are connected by long term memory (LTM) weights. An orienting subsystem O contains the vigilance parameter that resets F_2 if the recalled pattern is significantly dissimilar from the input pattern. A gray scale input pattern activation Ii enters the network at F_1, which eventually sends activation signals to F_2 via bottom up weights z_{ij}. These signals compete with the winning category in vector y_j, sending a recalled pattern via the top down weights z_{ji} at vector p_i. This activation vector, along with the short term memory (STM) activation, u_i, establishes a candidate model of the input pattern at r_i.

This expected activation vector is compared against the vigilance threshold ρ to determine closeness of fit. If the expected pattern sufficiently matches the current input, the STM vector is modified via vector v_i, and the system will resonate for the recalled category. If the expected vector is dissimilar, the orienting system will send an inhibitory signal to F_2, suppressing the winning category and allowing the network to test the next category.

The ART 2 architecture diagram shows the activation vectors for F_1 and F_2 and their interconnections. The black dots in the figure indicate the application of a non-specific excitation bias during the computation of the activation signal. The other arrows indicate the inputs to the vectors representing excitatory activities within the unit.

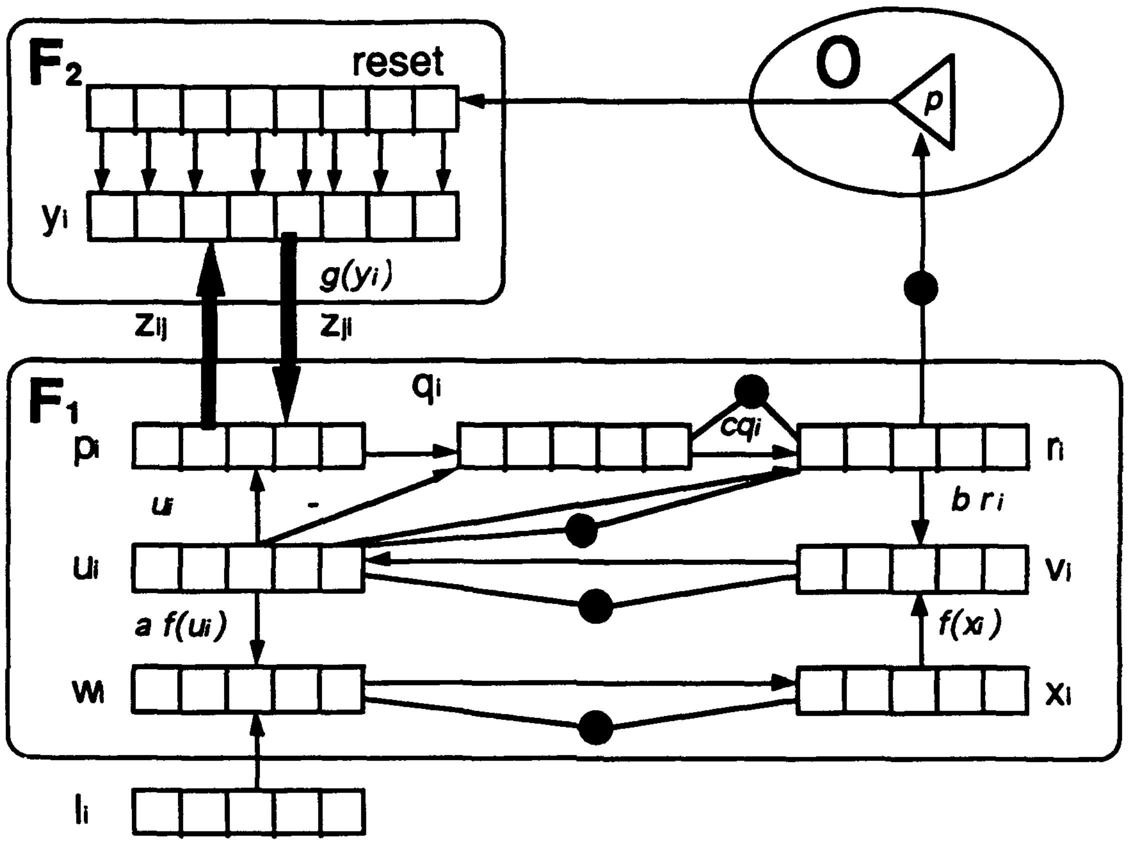


Figure 3-13 ART2 Architecture

ART 2 STM Equations: F1. The STM activity V_i of the ith element at any one of the F1 processing stages obeys the Lin Segal membrane equation:

$$\varepsilon \frac{d}{dt} V_i = -A V_i + (1 - B V_i) J_i^+ - (C + D V_i) J_i^- \qquad 3.2.8\text{-}12$$

for $i = 1 \ldots M$ where i is the number of elements in the input vector and J_i^+ is the total excitatory input to the ith element, and J_i^- is the total inhibitory input. With no input signal, the activity decays to 0 with a relax time given by A. The dimensionless parameter ε is the ratio between the STM relaxation time and the LTM relaxation time, which is $0 < \varepsilon << 1$. By setting $B = C = 0$, the ART 2 activation equation has the asymptotic form, where ε approaches 0, given by:

$$V_i = \frac{J_i^+}{A + D J_i^-} \qquad 3.2.8\text{-}13$$

In this form the dimensionless equations characterizing the STM activities, p_i, q_i, u_i, v_i, w_i and x_i are computed at F1 as follows:

$$p_i = u_i + \sum_j g(y_j) z_{ji} \qquad 3.2.8\text{-}14$$

$$q_i = \frac{p_i}{e + \|p\|}$$ 3.2.8-15

$$u_i = \frac{v_i}{e + \|v\|}$$ 3.2.8-16

$$v_i = f(x_i) + br_i$$ 3.2.8-17

$$w_i = I_i + au_i$$ 3.2.8-18

$$x_i = \frac{w_i}{e + \|w\|}$$ 3.2.8-19

where ‖V‖ denotes the inhibitory signal form the other elements within the unit V given by:

$$\|V\| = \sum_i^M \sum_j^M (V_i - V_j)^2$$ 3.2.8-20

and where y_j is the STM activity of the jth node in F2. The nonlinear signal function *f(x)* is of the form:

$$f(x) = \begin{cases} 0 & \textit{if}\ 0 \le x \le \vartheta \\ 1 & \textit{if}\ \ x > \vartheta \end{cases}$$ 3.2.8-21

which is piece-wise linear. The above activation equations, computed in the order given, will result in a dynamic network where the activations are shifted into the next activation as the input vector persists at the input.

ART 2 STM Equations: F2. Initially, F2 is inactive with no category selected from LTM. At this stage F2 is represented with an input signal from F1 via p_i which is an adapted form of the input pattern. The signal across all categories in LTM is computed concurrently as:

$$y_j = \sum_i p_i z_{ij}$$ 3.2.8-22

for $j = 1 \ldots N$ where N is the number of categories or storage capacity of LTM. When the signal passes a threshold and F2 makes a choice, the activation vector y_j is passed through a gated dipole threshold function given by:

$$g(y_J) = \begin{cases} d & \text{if } y_J = \max\{y_j \mid \text{the jth } F_2 \text{ node has not been reset on the current trial}\} \\ 0 & \text{otherwise} \end{cases}$$ 3.2.8-23

This signal is passed back to F1 via the top-down weights, z_{ji}, modifying the F1 activity at p_i which reduces to the following:

$$p_i = \begin{cases} u_i & \text{if } F_2 \text{ is inactive} \\ u_i + d z_{Ji} & \text{if the Jth } F_2 \text{ node is active} \end{cases}$$ 3.2.8-24

This activity is combined with the STM activation, u_i, to form the activation r_i. This last vector has properties essential to the orienting of the network.

ART 2 Orienting Subsystem Equations. The activity at r_i is such that $r_i$will attempt to model the input pattern with a pattern recalled from long term memory. That is to say, it will attempt to match the input pattern as given by the STM vector with a pattern which the system has previously learned and stored in LTM. The match may not be exact. The equation for this activity is:

$$r_i = \frac{u_i + cq_i}{e + \|u\| + \|cq\|}$$ 3.2.8-25

The activation vector has been normalized such that the sum of the square difference, $\|r\|$, will be 1 if the patterns match and will be less than one in proportion to the dissimilarity between STM and LTM patterns. The degree that patterns are allowed to be dissimilar before resetting the network is given by the vigilance parameter, ρ. The orienting subsystem will reset F2 if the following condition is satisfied:

$$\frac{\rho}{e + \|r\|} > 1$$ 3.2.8-26

where the vigilance parameter ρ is set between 0 and 1. The reset causes the previously winning node in y_j to be suppressed and then another pattern to be recalled from F2 and submitted to F1.

When the above condition is not satisfied, the network will begin to resonate. This can be observed in the activity of r_i. At first r_i will appear as a superposition of p_i and u_i, but then gradually becomes a modeled version of the input vector. It highlights those features of the input that categorize it in LTM. For high vigilance, r_i will become an accretive form of the input. For lower vigilance (i.e., more patterns within each category), specific features will be shown. This modifies the STM activity, u_i, which will show an idealized version of the input pattern.

When processing input patterns in the presence of noise, the non-specific parameter e acts as an excitation bias that desensitizes the network to noise fluctuations. In the computation of activation signals, e is the rate at which the activation decays given no excitatory or inhibitory input to the vector. In the computation of the vigilance ratio, e biases the reset to a higher vigilance threshold.

ART 2 LTM Equations. When the network resonates for a time scale that is long compared to the STM settling time, the LTM weights are modified by the following dynamic equations:

$$\frac{d}{dt}z_{Ji} = g(y_J)[p_i - z_{ji}] = d[p_i - z_{Ji}] = d(1-d)\left[\frac{u_i}{1-d} - z_{Ji}\right] \qquad 3.2.8\text{-}27$$

$$\frac{d}{dt}z_{iJi} = g(y_J)[p_i - z_{ij}] = d[p_i - z_{iJ}] = d(1-d)\left[\frac{u_i}{1-d} - z_{iJ}\right] \qquad 3.2.8\text{-}28$$

$$\forall j \neq J, \frac{d}{dt}z_{ji} = 0 \text{ and } \frac{d}{dt}z_{ij} = 0 \qquad 3.2.8\text{-}29$$

Adaptive Resonance Theory - Hierarchical (ART 3)

ART 3 completes the trilogy of adaptive resonance theory network architectures by implementing hierarchical search within multiple ART 2 networks. To achieve this, the F1 and F2 modules in the ART 2 architecture must become homologous and bi-directional. Note that F1 and F2 are not the same in the ART 2 architecture. In the ART 3 architecture, the F2 Field, which gives the category encoding of the input pattern at F1, is implemented as a F1 STM field. This means that the input pattern to F2 is homologous to the input pattern to F1. As a result, partial compression of the category encoding is introduced (ART 1 and ART 2 implement maximal compression encoding) to allow for multiple winners during the categorization of a dynamic input pattern. To allow for distributed competion between F1 bottom-up and F2 top-down retrieval, a medium term memory (MTM) is added to the the ART 2 architecture that models the chemical transmitters within biological neural systems. The MTM is longer scale to STM but sufficiently shorter time scale to LTM to allow the system to stablize before learning. These changes to the ART 2 allow the F1 STM fields to be cascaded within a multiple layer architecture where the STM represents both input patterns and partially compressed categories.

ART 3 Architecture. The architecture for ART 3 is shown in Figure 3-15. The sets of nodes, Fa, Fb, and Fc consist of the three layer unit architecture used in F1 of the ART 2 architecture. The output from Fa is input to Fb without adaptive weights. As a result the number of elements in the output from Fa matches the number of elements in the input of Fb. Adaptive bottom-up and top-down weights connect Fb and Fc. This allows the number of elements into Fc to differ from the number of elements from Fb. The orienting subsystem combines the STM output from Fa and Fb to establish the resonance pattern ri, which is compared to the vigilance parameter, r. The reset signal is sent to Fb and Fc and the to the adaptive weights connecting these modules.

By convention, $x_i^{\alpha\lambda}$ represents the input of the ith element in λth layer of the αth STM field, and $y_i^{\alpha\lambda}$ represents the output from the ith element in the λth layer of the αth STM field. The pairing of $x_i^{\alpha\lambda}$ and $y_i^{\alpha\lambda}$ defines a unit layer in the ART 3 STM field. This parallels the layering found in F1 of the ART 2 F1 STM field. The output from the middle layer represents the internal STM value for the input pattern and the output from the third layer is pattern that is effected by the combination of top-down retrieved and internal STM patterns.

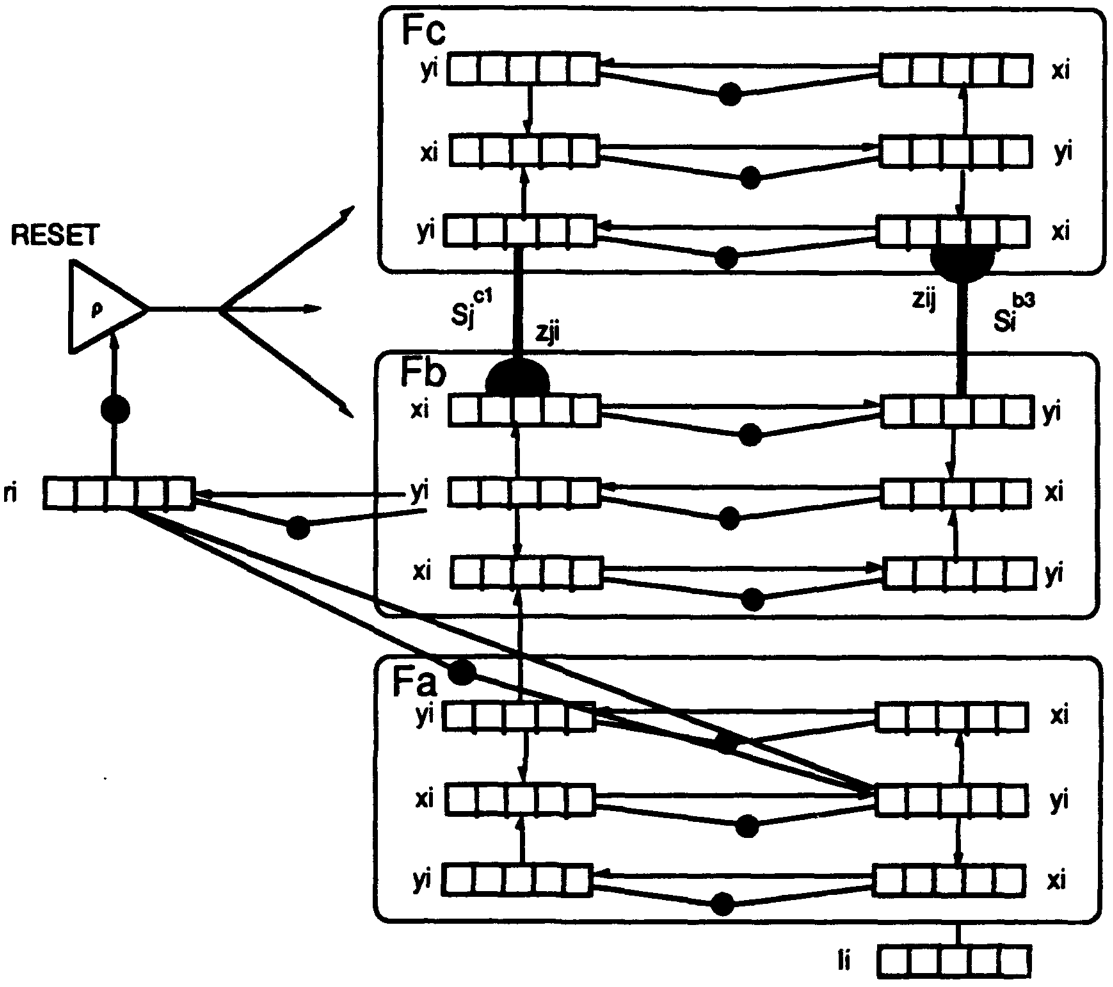


Figure 3-15 ART 3 Architecture

The LTM memory traces are as defined in the ART 2 architecture. However, in ART 3 the retrieved bottom-up and top-down signals are modulated over time by the chemical transmitters accumulating at each ith or jth node. The dynamics of the transmitters is given by the equations for presynaptic and bounded transmitters at each node. The presynaptic transmitters define the potential for generating chemical transmitters from a synapse node for a given signal and weight. The bounded transmitters define the resulting transmitters that are bound at the receptive synapse node. This results in a post synaptic activation that represents the input to the receiving STM layer. As a result of this mechanism, distributed nodes (i) can effect the state at a receptive node (j) over a time scale that is long compared to STM dynamics but is short with respect to LTM adaption.

The new ART search mechanism has a number of useful properties:

a. works well for mismatch, reinforcement,or input reset;

b. is simple;

c. is homologous to physiological processes;

d. fits naturally into network hierarchies with distributed codes and slow or fast learning;

e. is robust in that it does not require precise parameter choices, timing, or analysis of classes of inputs;

f. requires no new anatomy, such as new wiring or nodes, beyond what is already present in the ART 2 architecture;

g. brings new computational power to the ART systems;

h. although derived for the ART ꞏystem can be used to search other neural network architectures as well.

ART 3 STM Equations. Except during reset, equations used to generate the STM values are similar to ART 2 equations. Dynamics of the fields Fa, Fb, and Fc are homologous. The steady-state variables for the fields, when reset signal equals 0, are given by the following:

Input variable. The input value for the ith unit in the λth layer of the αth field is defined by the dynamics:

$$\varepsilon \frac{d}{dt} x_i^{\alpha\lambda} = -x_i^{\alpha\lambda} + S_i^{\alpha(\lambda-1)} + p_1^{\alpha\lambda} S_i^{\alpha(\lambda+1)} \qquad 3.2.8\text{-}30$$

In steady state,

$$x_i^{\alpha\lambda} \cong S_i^{\alpha(\lambda-1)} + p_1^{\alpha\lambda} S_i^{\alpha(\lambda+1)} \qquad 3.2.8\text{-}31$$

where l is 1, 2, or 3 and α is field a, b, or c. If l - 1 == 0, then α is the next field down from α, and l = 3. When l +1 == 4, α is the next field up from α and l = 1.

Output variable. The output variable is a normalization of the corresponding input variable. The output variable corresponding to the input variable for the ith unit in the λth layer of the αth field is defined by the dynamics:

$$y_i^{\alpha\lambda} \cong \frac{x_i^{\alpha\lambda}}{p_2^{\alpha\lambda} + \left| x^{\alpha\lambda} \right|} \qquad 3.2.8\text{-}32$$

where the interfield input signals are given by the non-linear signal function:

$$S_i^{\alpha\lambda} \equiv g^{a}\left(y_i^{\alpha\lambda}\right) \qquad 3.2.8\text{-}33$$

and the top-down intrafield signal across the adaptive synapse is:

$$S_i^{(\alpha+1)1} \equiv \sum_j v_{ji}^{(\alpha+1)\alpha} \qquad 3.2.8\text{-}34$$

The normalization is the Euclidean norm or L2 norm used in ART 2 (eq. 3.2.8-). This supports the orderly pattern transformations under a variable processing load and direct access to learned category representations without searching in LTM.

ART 3 Signal Functions. The ART 3 signal functions have two forms depending upon whether the matching is distributed (partially compressed) or choice (maximally compressed) encodings of the output variable:

Distributed

$$g^\alpha(w) = \begin{cases} 0 & \text{if } w \le p_7^\alpha + p_8^\alpha \\ \left(\dfrac{w - p_7^\alpha}{p_8^\alpha}\right) & \text{if } w > p_7^\alpha + p_8^\alpha \end{cases} \qquad 3.2.8\text{-}35$$

Choice

$$g^\alpha(w) = \begin{cases} 0 & \text{if } w \le p_7^\alpha \\ \left(\dfrac{w - p_7^\alpha}{p_8^\alpha}\right)^2 & \text{if } w > p_7^\alpha \end{cases} \qquad 3.2.8\text{-}36$$

In the case of choice the p7a parameter is dependent upon the number of categories by: $p_7^c = 1/\sqrt{n_c}$ so that the resulting output signal will reflect only one choice regardless of the number of choices. Otherwise the parameters p7 and p8 are non-negative (0.0 - 1.0).

ART 3 Transmitter Equations. When the reset signal equals 0, the levels of presynaptic and bound transmitter are governed by the following equations:

Presynaptic transmitter, Fb -> Fc

$$\frac{d}{dt}u_{ij}^{bc} = \left(z_{ij}^{bc} - u_{ij}^{bc}\right) - u_{ij}^{bc}\, p_5^c\left(x_j^{c1} + p_6^c\right)S_i^{b3} \qquad 3.2.8\text{-}37$$

Bound transmitter, Fb -> Fc

$$\frac{d}{dt}v_{ij}^{bc} = -v_{ij}^{bc} - u_{ij}^{bc}\, p_5^c\left(x_j^{c1} + p_6^c\right)S_i^{b3} \qquad 3.2.8\text{-}38$$

Presynaptic transmitter, Fc -> Fb

$$\frac{d}{dt}u_{ji}^{cb} = \left(z_{ji}^{cb} - u_{ji}^{cb}\right) - u_{ji}^{cb}\, p_5^b\left(x_i^{b3} + p_6^b\right)S_j^{c1} \qquad 3.2.8\text{-}39$$

Bound transmitter, Fc -> Fb

$$\frac{d}{dt} v_{ji}^{cb} = -v_{ji}^{cb} - u_{ji}^{cb} p_5^b \left(x_i^{b3} + p_6^b \right) S_j^{c1} \qquad 3.2.8\text{-}40$$

ART 3 Reset Equations. Reset occurs when patterns active at Fa and Fb fail to match according to the criterion set by the vigilance parameter. The reset unit for the ith node is

$$r_i^b \cong \frac{y_i^{a2} + y_i^{b2}}{p_3^a + \left\| y^{a2} \right\| + \left\| y^{b2} \right\|} \qquad 3.2.8\text{-}41$$

Reset of the Fb and Fc fields occurs if

$$\left\| r^b \right\| < \rho^b \qquad 3.2.8\text{-}42$$

where

$$0 < \rho^b < 1 \qquad 3.2.8\text{-}43$$

The effect of a large reset signal is approximated by setting input varables and bound transmitter variables equal to 0.

3.2.9 Brain State in a Box Function Blocks (CSC09)

CSC Purpose. This CSC provides custom function blocks for implementing and training Brain State in a Box (BSB) neural networks within embedded system designs. The CSC includes custom coded function blocks for unsupervised learning and recall processing algorithms. It also provides custom user function blocks which allow ART networks to perform various functions within a communications design.

Execution and control data flow. The execution and control data flow is defined by the propagation architecture of Brain State in a Box (BSB). The following defines the overall architecture to be implemented by this CSC.

BSB: Linear Associator. The BSB Network is an example of associative memory , which has been extensively investigated by Hopfield and Kosko. It is based upon the properties of a linear associator using a generalized Hebb learning rule. The output, y_j, from the linear associator is generated from the weighted input, x_i, as follows:

$$y_j = \sum_i w_{ij} x_i \qquad 3.2.9\text{-}1$$

where i = 1 ... N and j = 1 ... M where N and M are the number of input and output elements respectively. The associative memory weights, w_{ij}, are adjusted according to a generalized Hebb rule with each input/output pair, k, given by:

$$\delta w_{ij} = \alpha y_j^k x_i^k \qquad 3.2.9\text{-}2$$

where y_k and x_k are the kth associative learning example. The associative weights are the sums of the output vector products,

$$w_{ij} = \eta \sum_{k=1}^{n} y_j^k x_i^k \qquad 3.2.9\text{-}3$$

where η is a learning constant.

To illustrate the properties of this network, define the input as a combination of a mean, p, combined with a non-linear distortion, d^k, as follows:

$$x_i^k = p_i + d_i^k \qquad 3.2.9\text{-}4$$

Substituting this into equation 3.2.9-3, the associative weights are:

$$w_{ij} = \eta \sum_{k=1}^{n} y_j x_i^k = \eta y_j \left(n p_i + \sum_{k=1}^{n} d_i^k \right) \qquad 3.2.9\text{-}5$$

where n is the number of training examples, x^k, and y is the corresponding output state associated with these examples. If it is assumed that the sum of d^k has a zero mean error, then the associative weights, relate the output, y_j, with the mean input, p_i, given by:

$$w_{ij} = \eta n y_j p_i \qquad 3.2.9\text{-}6$$

Even though none of the examples explicitly provide the mean, the network is able to extract the mean input even if the distortion is non-gaussian or non-linear.

BSB Architecture. Figure 3-16 gives the architecture for a general application of the above linear associator. The outputs for the BSB processing elements are fully connected to one another, such that the output state s_j of the jth unit at time t is given by:

$$s_j(t) = \alpha \sum_i w_{ij} y_i(t) + \gamma y_j(t) + \delta x_j(0) \qquad 3.2.9\text{-}7$$

The first term, passes the current system state, $x_i(t)$, through the weight connections to generate the associative state, y_j at the current time step. The second term causes the current output state to decay slightly. This has the effect of forcing the distortion to zero over multiple interactions. The third term keeps the initial information constantly present and has the effect of limiting the flexibility of the possible states of the dynamical system since some vector elements are strongly biased by the initial input. The output is passed through a non-linear limit function to generate the next state of the system:

$$y_j(t+1) = f\left(s_j(t)\right) \qquad 3.2.9\text{-}8$$

The purpose of the limit function, f(), is to maintain the output state within limits for multiple interactions of feedback through the network. The output is not allowed to exceed the positive limit or be less than the negative limit. The limiting process contains the state vector for the dynamical system, hence the designation, brain state in a box.

The dynamical system is free to move within the boxed limits established by the network. For multiple iterations, the state fluctuates until it settles to a stable state for the given initial input vector. For auto-associative, symmetrical weights the final state is associated with the minimum energy state of the network. By changing the input state, the system will move to another attractor which has been learned by the network.

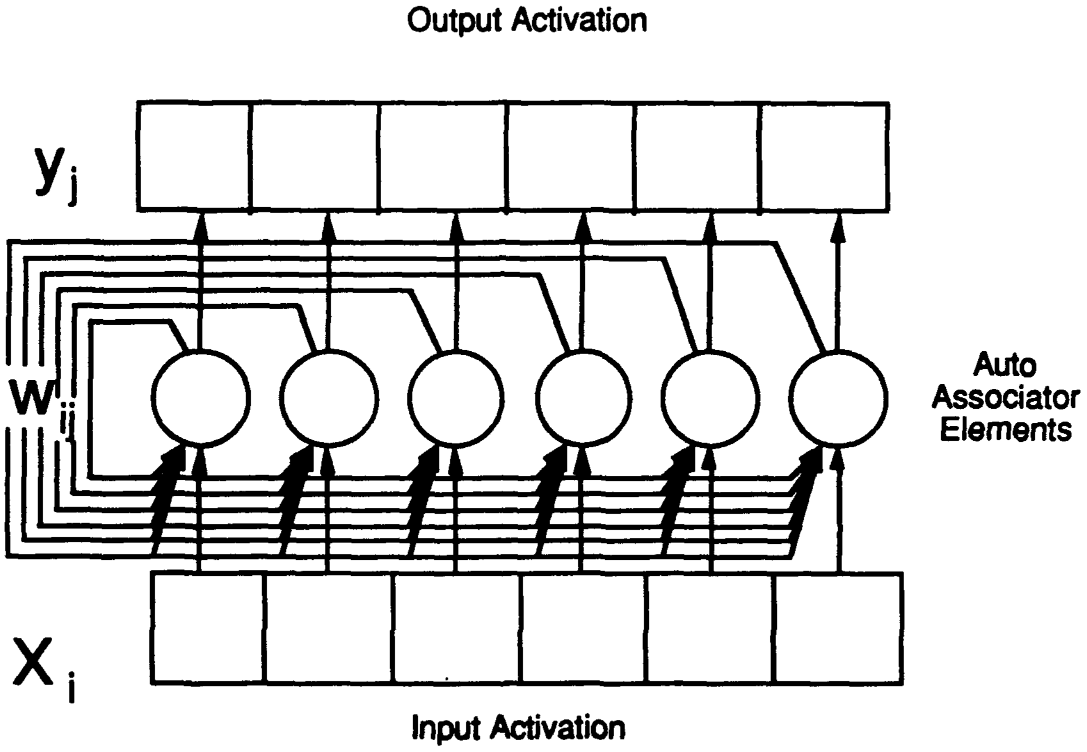


Figure 3-16 Brain State in a Box Architecture

BSB: Error Correction Equation. By using a Widrow-Hoff error correcting procedure, the associative weights are incrementally adjusted by Δw given by:

$$\Delta w_{ij} = \eta\left(y_j^k - \sum_l w_{lj} x_l^k\right) x_i^k \qquad 3.2.9\text{-}9$$

This error correcting procedure will have the accumulative effect of forcing error to zero so that the weights approximate a least means squares mapping of the input to the output vector.

4. NEURAL NETWORK COMMUNICATIONS SYSTEM APPLICATIONS

This section summarizes the results of simulating Neural Network applications in Communications Signal Processing. The simulation objectives were:

1. Demonstrate a Neural Network simulation capability for the study of Neural Network applications to communications systems and signal processing.

2. Investigate how selected areas of communications can benefit from Neural Network technology via the developed simulation capability.

3. Use the simulation to identify neural network configurations to be included in the conceptual design of a phase II neural network transceiver.

In the following summary of selected applications of Neural Network technology, each simulation is summarized using the following format:

Introduction-	A brief statement to preface the simulation which follows.
Overview-	A description of the problem, including a block diagram of the system and background information of certain system components.
Simulation Parameters-	Specific parameters used by the Neural Network configuration.

Paradigm	Neural Network architecture/learning algorithm used
Input Nodes	Size of the input vector
Hidden Nodes	Number of internal Neural Processing nodes
Output Nodes	Number of output nodes (the size of the output vector)
Learn Rate	Degree to which the error signal is applied to weight changes
Momentum	Degree to which previous weight changes influence future weight changes
Update Interval	Number of training vectors between weight updates
Number of passes	Number of times that the training set was applied during learning
Training Size	The total training size, giving the total number of input vectors applied during learning
SPW Iterations per vector	The number of simulation iterations to produce a single input vector

Results-	Observations and data collected which displays the results of the Neural Network, including performance curves and signal plots.
Lessons Learned-	Experience gained in the effort to apply the Neural Network technology to the problem which may be useful in other Neural Network applications. (Included when applicable.)
Potential Extension-	A discussion of areas of future research for Neural Network applications related to the current problem. (Included when noteworthy.)
Procedures-	Procedures to execute the simulations described in Section 4 are documented in the **Appendix**.

4.1 SIMPLE NON-LINEAR MAPPING WITH A NEURAL NETWORK

Application

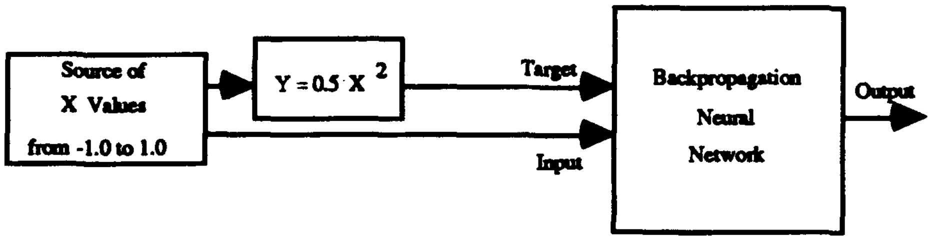
Non-Linear Mapping by a Backpropagation Neural Network

Introduction

Backpropagation Neural Networks have been shown to be capable of emulating continuous functions and discrete mappings from one input vector to another. In an initial experiment, we investigated the capability of Backpropagation networks to learn a simple non-linear mapping such as the square function.

Overview

This problem was studied using the block diagram below:



A training set was created consisting of 21 real numbers between -1.0 and 1.0, inclusive, spaced by a distance of 0.1. The target consisted of the input value squared, multiplied by a factor of 0.5. The factor 0.5 was chosen because the output of the Backpropagation network produces values ranging from

-0.5 to 0.5 (which is a result of the sigmoid non-linear transfer function in each output node). An input value and target value were presented to the Backpropagation Network at each iteration. Weights were updated in batch mode after each pass through the training set. In a second experiment, the neural network was trained to learn the $0.5x^3$ function.

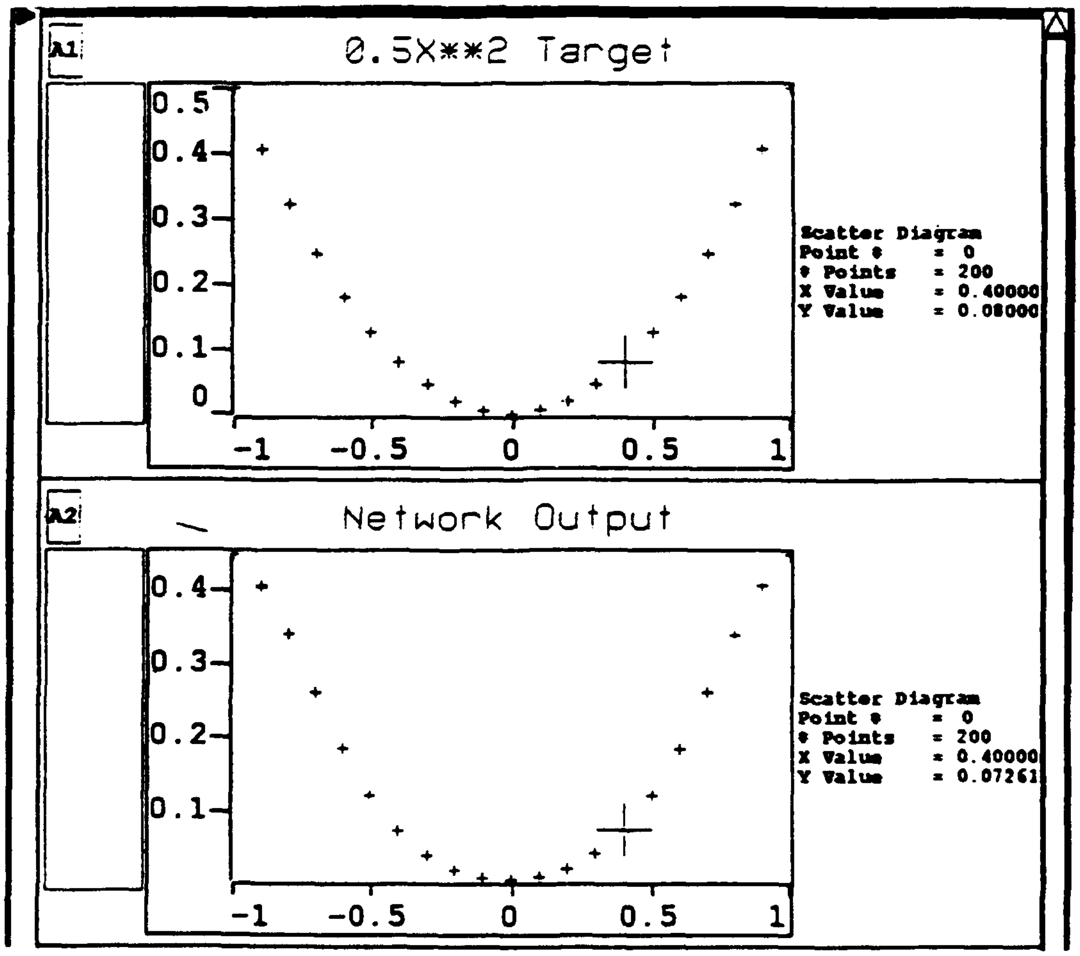
Simulation Parameters

Paradigm	Backpropagation
Input Nodes	1
Hidden Nodes	9
Output Nodes	1
Learn Rate	1.5

Momentum	0.9
Update Interval	21
Number of Passes	476
Training Size	9996
SPW Iterations per Vector	1

Results

Figure 4.1-1 displays two X vs Y plots where the neural network is trained to learn the $0.5x^2$ function. Plot A1 shows the target value (vertical axis) for each corresponding input value (horizontal axis) in the training set. Plot A2 shows the mapping created by the neural network after training has completed.



```
Scatter Diagram
Point #    = 0
# Points   = 200
X Value    = 0.40000
Y Value    = 0.08000
```

```
Scatter Diagram
Point #    = 0
# Points   = 200
X Value    = 0.40000
Y Value    = 0.07261
```

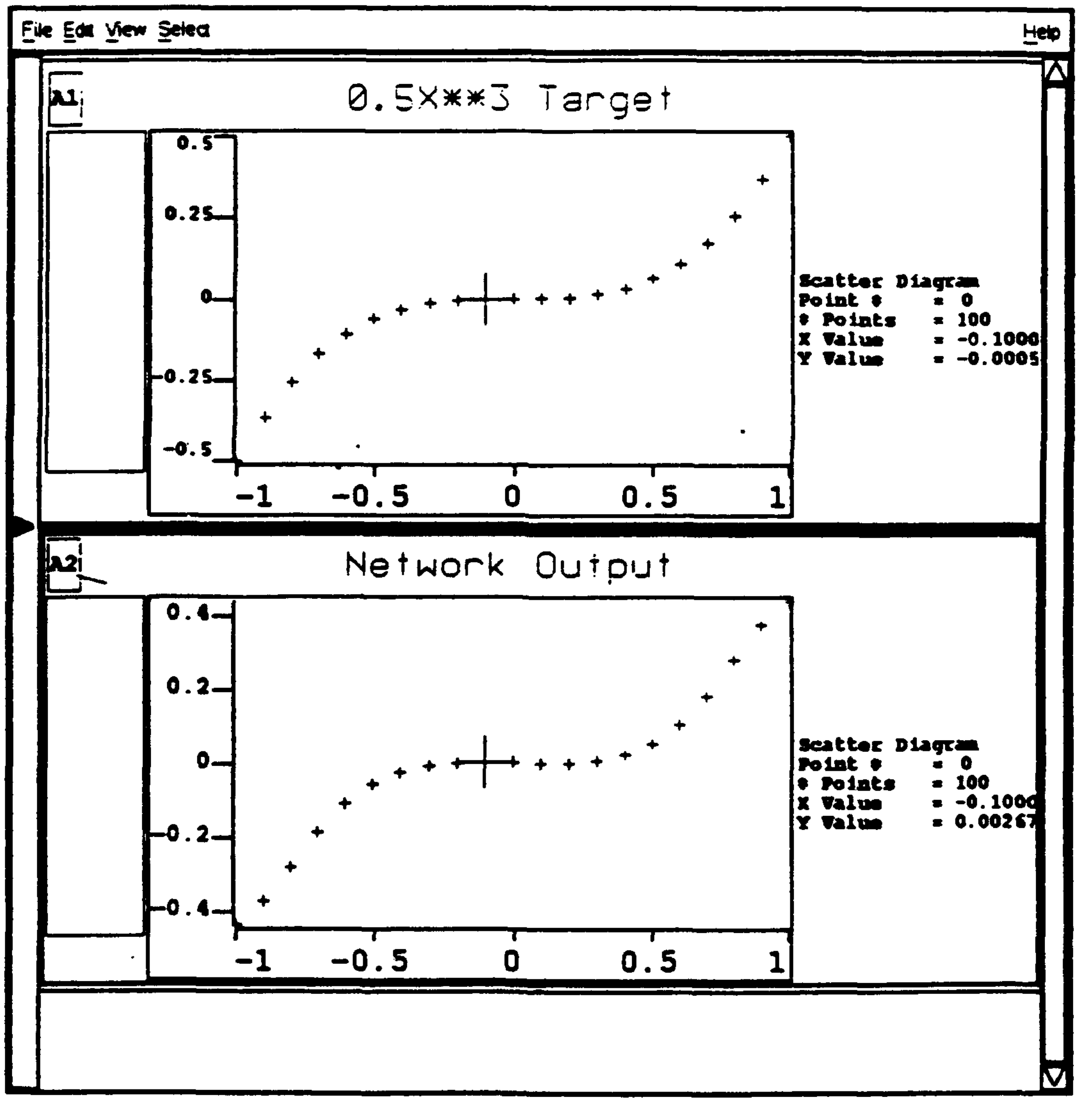
Figure 4.1-1 Neural Network Target and Output for $0.5x^2$ Function

Figure 4.1-2 displays similar plots for the $0.5x^3$ function. Further training with a lower learning rate would increase the precision of the mappings.

The neural network has been trained to learn a function by providing it with only 21 pairs of input and target values over the range from -1.0 to 1.0. To examine the ability of the network to generalize, i.e., perform the desired mapping for inputs for which it has not been trained, we held the weights at their final values and added noise to the input values, thus approximating a continum of inputs. Figure 4.1-3 shows that the network can indeed generalize, which in this case means specifically that the neural network effects a smooth interpolation between the trained points. However, the A2 plot in Figure 4.1-3 suggests that the generalization is not valid for values of x outside the interval from -1 to 1. Note how the A2 plot in Figure 4.1-3 reveals more effectively than A1 the incomplete convergence to the square function, even though the A2 plot overlays the A1 plot.

Lessons Learned

Upon initialization of the Backpropagation algorithm, weights are initialized randomly between an upper and lower bound. As a default, -0.3 and +0.3 are used. These bounds produced inferior results. For the square function, training time was significantly longer. The mapping remained linear for about 400 passes through the training set before significant convergence began. For the $0.5x^3$ function, a local minimum was found and the correct maping was never reached. Upon changing the initial bounds to -4.0 and 4.0, the network quickly converged to a good solution.



```
Scatter Diagram
Point #      = 0
# Points     = 100
X Value      = -0.1000
Y Value      = -0.0005
```

```
Scatter Diagram
Point #      = 0
# Points     = 100
X Value      = -0.1000
Y Value      = 0.00267
```

Figure 4.1-2 Neural Network Target and Output for 0.5x^3 Function

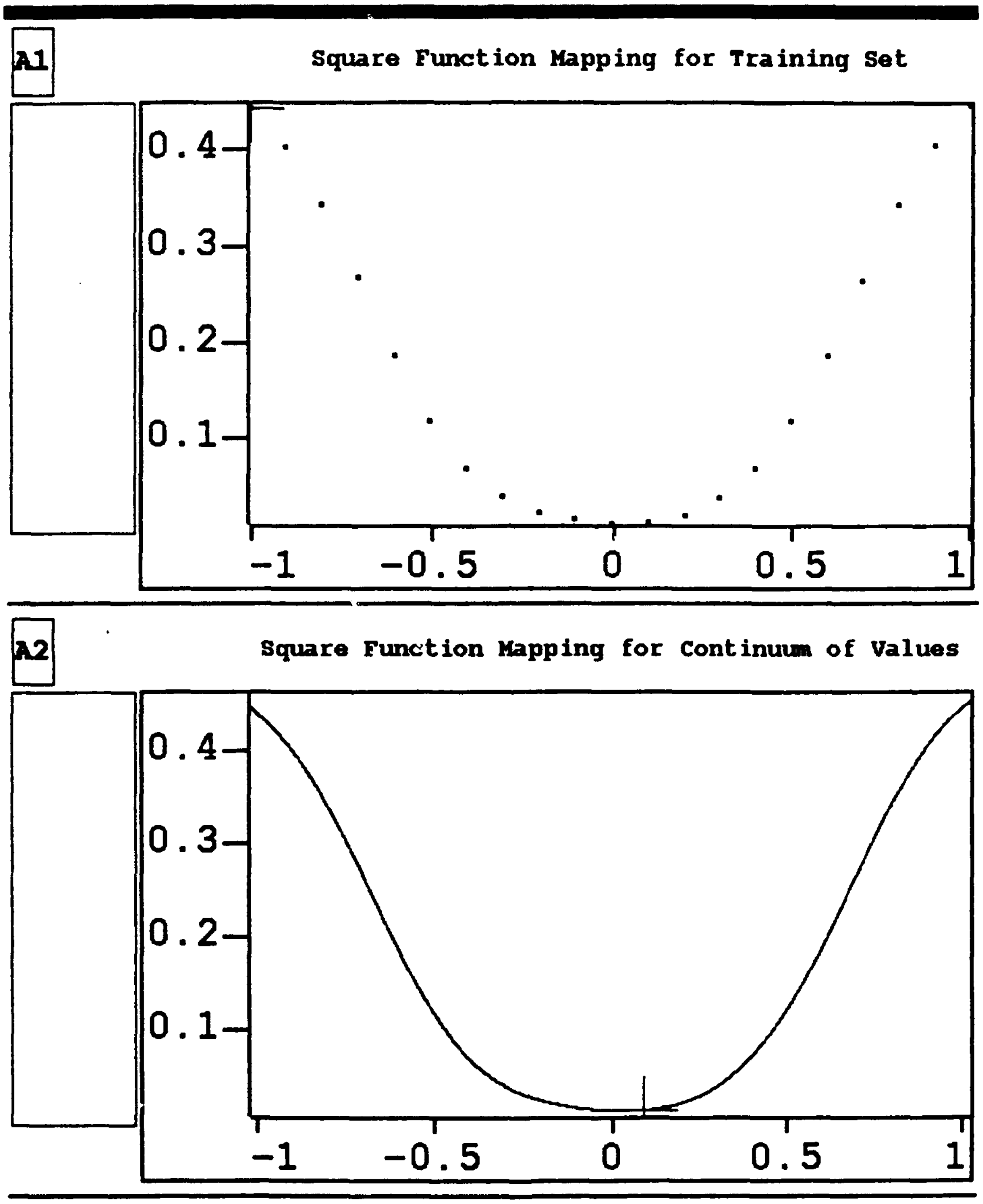


Figure 4.1-3 Trained and Generalized Outputs for the $0.5x^2$ Target Function

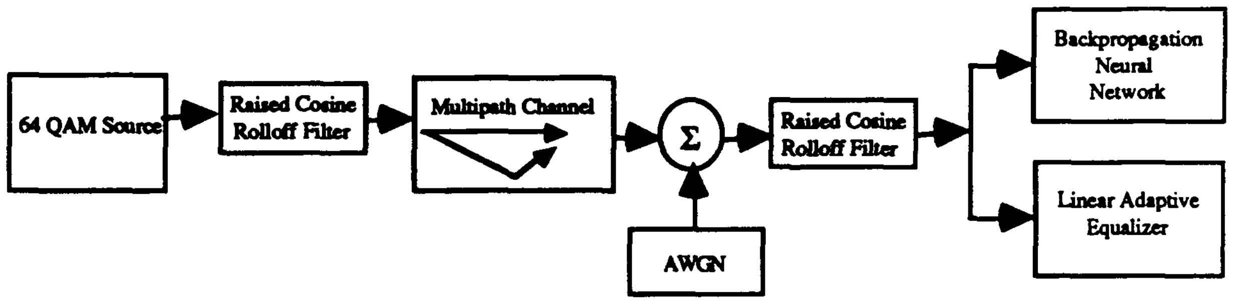
4.2 EQUALIZATION OF MULTIPATH DISTORTED 64-QAM

Application

Equalization of Multipath Distorted 64-QAM using a Backpropagation Neural Network

Introduction

The use of Neural Networks in the equalization of signals exhibiting intersymbol interference (ISI) was studied using the block diagram below:



64 QAM Source
Raised Cosine Rolloff Filter
Multipath Channel
Σ
AWGN
Raised Cosine Rolloff Filter
Backpropagation Neural Network
Linear Adaptive Equalizer

Overview

A transmitter generating random 64-QAM symbols at a sampling rate of 20 samples/symbol was used as the system input. A 256-tap raised-cosine rolloff filter with a rolloff factor of 0.5 was used for pulse shaping at the transmitter side. The channel model used was a multipath Rummler model consisting of the primary signal summed with a delayed, rotated, and attenuated version of the primary signal. Additive white Gaussian noise (AWGN) with a variance of 0.005 was added to the multipath signal to complete the channel model. A 256-tap raised-cosine rolloff filter with a rolloff factor of 0.5 was also used for pulse shaping at the receiver side, fulfilling Nyquist's criteria for minimizing ISI. The ISI-distorted signal (after receiver pulse-shaping) is input to a Linear Adaptive Equalizer and a Backpropagation Neural Network for comparison. The channel model causes ISI to occur, necessitating the use of equalization for proper demodulation. In the Backpropagation Neural Network and the linear equalizer, one in-phase sample and one quadrature-phase sample of each of the last 16 received symbols are used as input, hence 32 inputs. The output layer produces an in-phase and quadrature pair which represents the equalized signal at the optimum sampling time for demodulation.

Rummler Channel

The Rummler channel model consists of the primary signal summed with a delayed, rotated, and attenuated version of the primary signal. The input to output relationship for this channel is:

$$y(t) = x(t) - \beta x(t-\tau)e^{j2\pi\tau f_0}$$

where $x(t)$ is the Rummler channel input

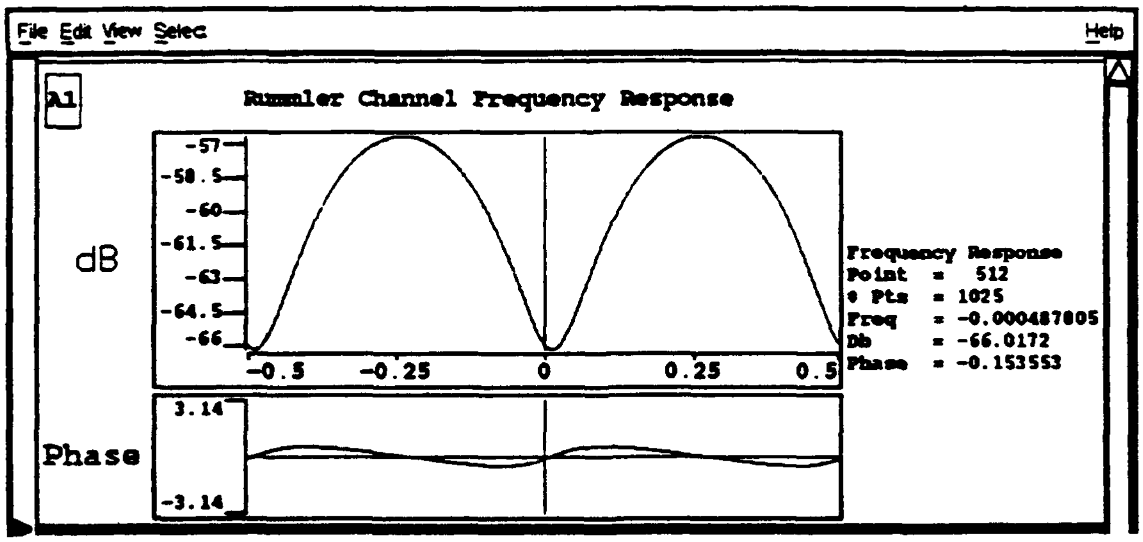
$y(t)$ is the Rummler channel output

β is the gain of the reflected signal

τ is the time delay between primary and reflected signals

f_0 is the null frequency

Taking a Fourier transform of the Rummler channel impulse response gives a spectrum with severe attenuation for certain frequencies, called nulls (see Figure 4.2-1). The Rummler channel model lets the user specify these frequencies. The null frequency used in this system is 4 MHz. The sampling frequency is 320 MHz. The baud rate is 16 MHz.



```
Frequency Response
Point   =   512
# Pts   =  1025
Freq    =  -0.000487805
Db      =  -66.0172
Phase   =  -0.153553
```

Figure 4.2-1 Rummler Channel Frequency Response

Raised Cosine Rolloff Filters

The raised cosine rolloff filter is a family of lowpass filters which assist in satisfying Nyquist's First Method for eliminating Intersymbol Interference (ISI). Basically, the goal is to have a system with equivalent transfer function

$H_e(f) = H(f)H_{tx}(f)H_c(f)H_{rx}(f)$,

where $H(f)$ is the frequency spectrum of the modulated signal before filtering

$H_{tx}(f)$ is the transfer function of the Transmitting filter

$H_c(f)$ is the transfer function of the Channel

$H_{rx}(f)$ is the transfer function of the Receiving filter

such that the corresponding system impulse response, $h_e(t)$, is zero at all sampling times other than the time associated with the transmitted symbol at that instant:

$h_e(kT_s)$ $= C$ for k=0,

$= 0$ for k not equal to 0

where T_s is the sampling period

C is the value of the transmitted symbol

If this is true, the adjacent symbols in a stream will not bleed into each other ***at the sampling times***, which are the only times of interest. Choosing a raised cosine rolloff filter for $H_{tx}(f)$ and $H_{rx}(f)$ with a well-behaved channel will satisfy Nyquist's First Method. However, the channel may not be well-behaved or it may even be varying. An equalizer (or Neural Network) can be used to compensate for the channel behavior in order to result in a total system transfer function, $H_e(f)$, satisfying the above criteria.

This Raised Cosine Rolloff Filter block performs frequency-domain filtering. This routine is typically much faster than the time domain implementation which uses a linear convolution method. The generated raised-cosine frequency response can be either a complete raised-cosine or a square-root raised-cosine filter depending on a parameter. Also another string parameter is provided if a complete (or square-root) raised-cosine cascaded with an inverse sinc function is required. The actual filtering is implemented in the following steps: The input vector signal is zero padded so that its length is twice as large. That is, the length of the zero-padded input signal is equal to the number of interpolation points (or FIR tap length). The complex FFT of this signal is then taken, and the resulting vector weighted by the frequency domain description of the filter which was calculated during initialization. Then the inverse FFT is taken to convert the resulting sequence back to the time domain The output is then delayed and summed to realize the "overlap and add" computation of the output.

Network Parameters

Paradigm	Backpropagation
Input Nodes	32
Hidden Nodes	2
Output Nodes	2
Learn Rate	0.3
Momentum	0.9
Update Interval	20*
Number of Passes	2500
Training Size	50K
SPW Interations per vector	20

*The training set consisted of a continuous stream of random 64 QAM received symbols and the corresponding target symbols. Weights were updated at the end of every 20 input vectors. The I and Q values for the last 16 received symbols constituted the input vector.

Results

In Figure 4.2-2 Signal A1 represents the constellation for the equalized, unquantized signal at the output of the linear equalizer. Signal A2 shows the same for the Neural Network. Signal A3 displays the rotated, ISI distorted signal constellation appearing at the input to the Neural Network and the equalizer. Both the Neural Network and the linear equalizer were able compensate for the majority of the distorting effects of the multipath channel.

Both the Neural Network and the linear adaptive equalizer exhibit the following functionality:

1. Both can correct intersymbol interference with "multipath delay" on the order of several symbols (e.g. 30 samples) when given a training signal. We are using a Rummler channel which consists of a primary signal component summed with a scaled, rotated, and delayed secondary component. The amount of this delay in time is what we are specifying when we refer to the "multipath delay". This delay is specified to be some number of samples. To determine a delay in seconds, you must know the sampling rate, f_s. Using our terminology, one sample of multipath delay equals $1/f_s$ seconds.

2. Both can correct ISI with multipath delay on the order of a few samples (e.g. 2 samples) from start-up using hard decision feedback instead of a training signal, given a suitable initial weight setting. For the linear equalizer this suitable initial weight setting was to initialize all taps to 0.0 except for the center tap which was set to 1.0. For the Neural Network, the initial weights were set to those resulting from training to a zero delay multipath situation.

3. Both can continue to correct ISI with either a fixed multipath delay or a slowly varying multipath delay using hard decision feedback instead of a training signal, given prior convergence. Here, slowly varying means step increases (or decreases) of 1 sample of multipath separated by sufficient time to allow for re-convergence.

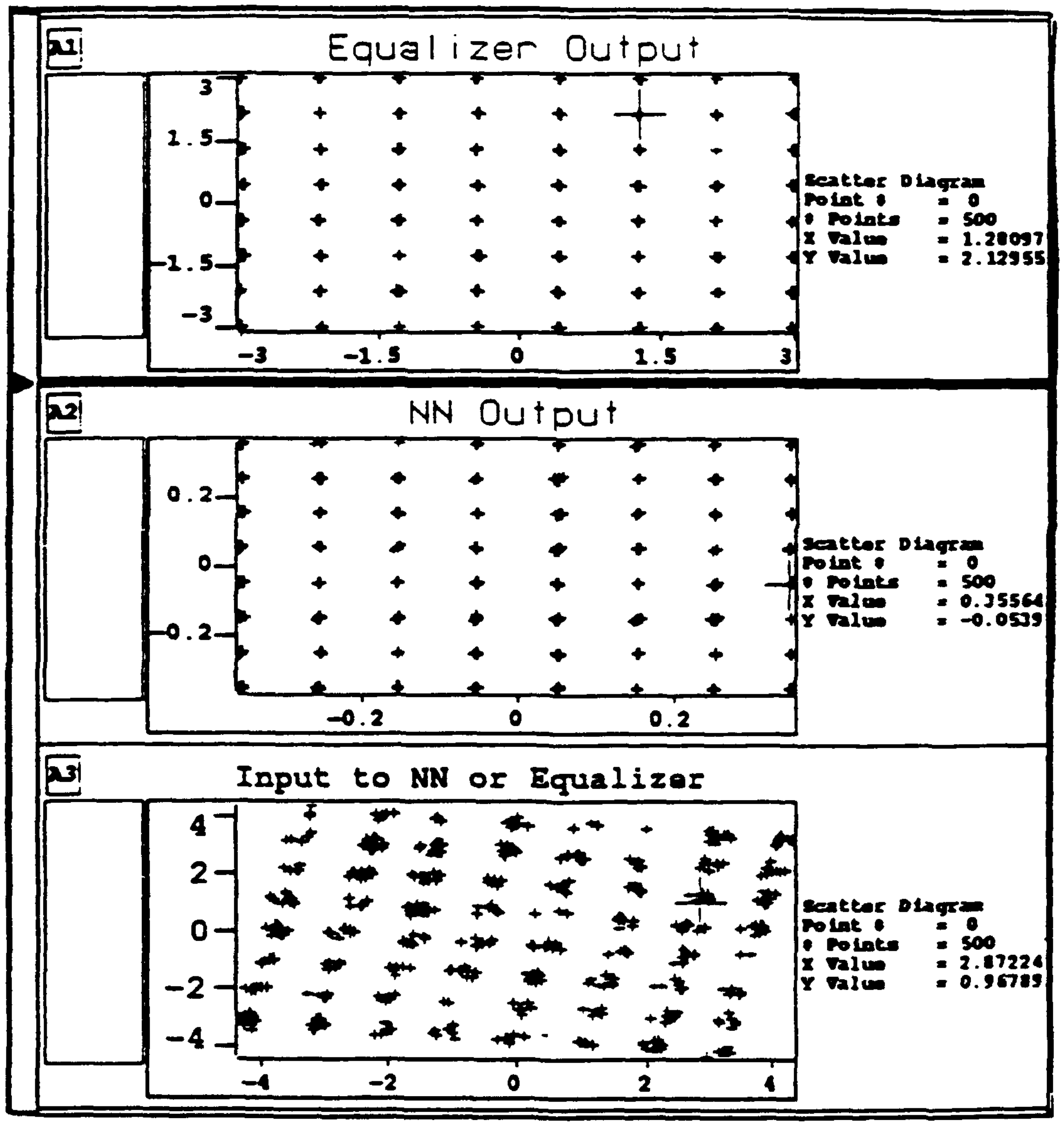
4. Both will not correct ISI for a rapidly changing multipath delay. Here, rapidly changing multipath delay means step increases (or decreases) of 5 or more samples .

Lessons Learned

Scaling of signals is very important for the Neural Network. Input and target signals must be scaled appropriately to remain within the bounds of the sigmoid. Improper scaling may cause saturation of the sigmoids which in turn will cause training to virtually cease (network paralysis) and/or output levels to be clipped at the upper and lower bounds of the sigmoid.

Synchronization and timing in SPW are also very important. Careful synchronization between input signals and corresponding target signals is required. An external clock is used to control the clocking of samples into the network. Only the center sample of each symbol is input to the Neural Network. Weights are updated once per symbol.

Convergence time for the Neural Network requires about 50 times as many iterations as the linear adaptive equalizer.



```
Scatter Diagram
Point #     = 0
# Points    = 500
X Value     = 1.28097
Y Value     = 2.12955
```

```
Scatter Diagram
Point #     = 0
# Points    = 500
X Value     = 0.35564
Y Value     = -0.0539
```

```
Scatter Diagram
Point #     = 0
# Points    = 500
X Value     = 2.87224
Y Value     = 0.96789
```

Figure 4.2-2 Signal Constellations for 64-QAM Equalization

4.3 EQUALIZATION OF DYNAMIC MULTIPATH DISTORTION

Application

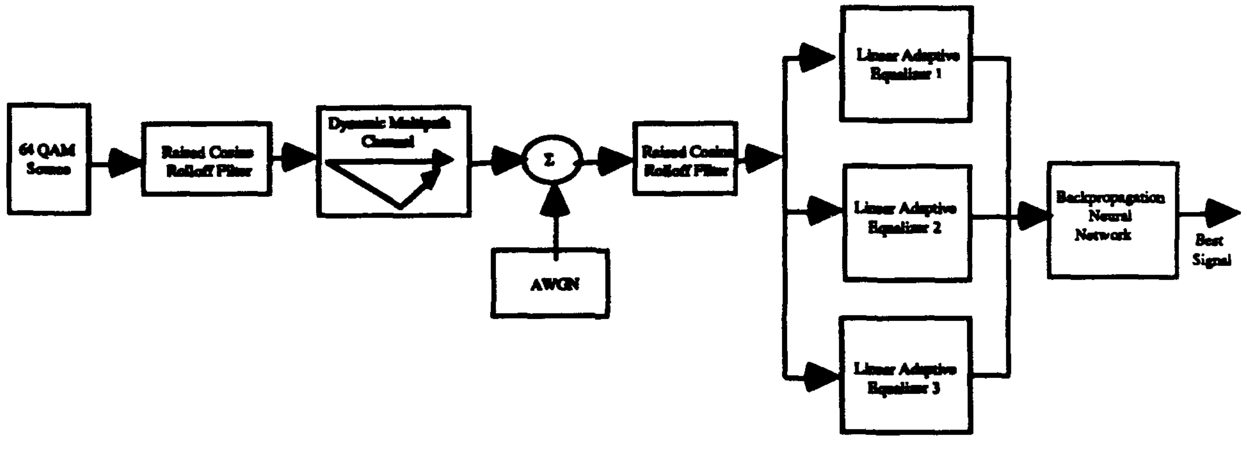
Use of a Backpropagation Neural Network to Control a Bank of Equalizers in a Dynamic Multipath Environment

Introduction

The simulation of the multipath-distorted 64 QAM system previously discussed showed that a Backpropagation Neural Network could equalize a rotated constellation, and also adapt to "slow" changes of the multipath delay in an unsupervised scenario (using its own output as a target). This was true of the Linear Adaptive Equalizer as well. However, if the multipath delay were to change suddenly and to a "large" degree, both the Neural Network equalizer and the linear adaptive equalizer would not be able to keep up and would require a training signal to once again cancel the effects of multipath. This section is a summary of the use of N equalizers (neural or linear adaptive), where each has been trained (and have their weights fixed) to a different amount of multipath delay, to create a structure which will be able to handle a large range of dynamic multipath distortion.

Overview

The use of Neural Networks in the equalization of signals exhibiting intersymbol interference (ISI) as a result of a dynamic multipath distortion was studied using a structure described by the block diagram below.


64 QAM Source
Raised Cosine Rolloff Filter
Dynamic Multipath Channel
Σ
AWGN
Raised Cosine Rolloff Filter
Linear Adaptive Equalizer 1
Linear Adaptive Equalizer 2
Linear Adaptive Equalizer 3
Backpropagation Neural Network
Best Signal

A transmitter generating random 64-QAM symbols was simulated using a sampling rate of 20 samples/symbol. A 256-tap raised-cosine rolloff filter with a rolloff factor of 0.5 was used for pulse shaping on the transmitter side. The channel model used was a multipath Rummler model consisting of the primary signal summed with a delayed, rotated, and attenuated version of the primary signal. The input to output relationship for this channel is:

$$y(t) = x(t) - \beta x(t-\tau)e^{j2\pi\tau f_0}$$

where x(t) is the Rummler channel input

y(t) is the Rummler channel output

β is the gain of the reflected signal

τ is the time delay between primary and reflected signals

f_0 is the null frequency

The time delay (τ) is varied to represent a dynamic multipath channel.

Additive white Gaussian noise with a variance of 0.005 was added to the multipath signal to complete the channel model.

A 256-tap raised-cosine rolloff filter with a rolloff factor of 0.5 was also used for pulse shaping on the receiver side. The channel model causes ISI to occur, necessitating the use of equalization for proper demodulation.

We are using a Rummler channel which consists of a primary signal component summed with a scaled, rotated, and delayed secondary component. In some practical cases, the delay of the secondary path relative to the primary path is a fixed, constant value. For example, a Line of Sight microwave link may have a secondary path due to the reflection off a nearby building. If the transmitter and receiver are stationary, the relative delay between the signal components due to the reflection is constant. In other situations, the relative delay may be dynamic due to something in the geometry of the system being in motion. In practice, situations like this can also occur when changing atmospheric conditions alter the path of one (or both) of the two main signal components thereby affecting the time delay between them. "Slowly" and "quickly" varying multipath are qualitative terms we used to gauge how fast the relative delay is changing. Here, "slowly varying" means step increases (or decreases) of one sample of multipath delay separated by sufficient time to allow for re-convergence. "Quickly varying" means step increases of more than one sample of multipath delay. As before, we equate one sample of multipath delay to a relative delay between the signal components of $1/f_s$ seconds.

The Backpropagation Neural Network is used to decide which of the fixed equalizers is best at equalizing the multipath distortion (in effect estimating the value of the dynamically varying multipath delay). It makes decisions based on a function of the last three complex symbols output from each equalizer. This yields an input vector of size 18: 2 elements per complex symbol for the last 3 symbols for each of 3 equalizers. To expedite training, we use the absolute value of the difference between the complex symbol output of an equalizer and the nearest 64 QAM symbol level as the inputs to the Neural Network. The output of the Neural Network is a vector identifying which of the equalizers is correctly cancelling the effects of multipath at the current time. As the multipath channel varies dynamically, the Neural Network will make decisions dynamically as to the proper equalizer to use for signal demodulation. This structure can be expanded to include more equalizers. Such a structure would be more robust and be able to compensate for a wider range of multipath fluctuation and in a faster manner compared to conventional methods.

Linear Adaptive Equalizers

Several 16-tap linear adaptive equalizers are connected to the output of the raised cosine filter at the receiver side. Each of these equalizers have been trained (and have their weights fixed) to a different amount of multipath. We have shown that Backpropagation Neural Networks are capable of performing linear equalization, and could be used here instead of the linear adaptive equalizers. We use the linear adaptive equalizers simply for convenience in this simulation.

The SPW equalizer block implements a minimum-mean-square error linear adaptive equalizer for QAM. It has an equalizer input, training sequence input, quantized QAM output, unquantized QAM output, and the tap weights output. Parameters include the number of taps, first angle, QAM order, feedback gain. The taps are separated by one symbol interval in time. The sample at symbol center is used to update the taps, i.e. this is assumed to be the point at which the eye is most open, and will be the point forced open by the equalizer. The feedback gain constant should be made smaller as the number of taps increases. If it is too large for the number of taps, then the equalizer may not converge. A control signal chooses either the decision feedback or the reference (training) signal for the feedback loop.

Generation of Training Data

Figure 2 depicts the SPW system which provides the input data for the Neural Network. Initially, each of the linear equalizers are themselves trained to a specific multipath delay (0, 2, and 4 samples of the 64 QAM signal). Once convergence is reached, the equalizer taps are fixed. At this point, a random number generator is used to specify a multipath delay of either 0, 2, or 4 samples. The resulting ISI-distorted signal is processed by each of the equalizers. For each received symbol, a vector of length 18 (as previously described) is written to disk.

In addition, a vector of length 3 (v0, v1, v2) is written to disk representing the target vector. For a given multipath delay (0, 2, or 4 samples), one of the 3 equalizers produces superior results in comparison to the others. A target vector is created as follows:

Equalizer Producing Superior Results	Target Vector
1	(0.5, -0.5, -0.5)
2	(-0.5, 0.5, -0.5)
3	(-0.5, -0.5, 0.5)

The process is repeated for each of 64 received symbols such that a training set of 64 input-target pairs is created.

Network Parameters

Paradigm	Backpropagation
Input Nodes	18
Hidden Nodes	3
Output Nodes	3
Learn Rate	0.2
Momentum	0.0
Update Interval	64

Number of Passes	1719
Training Size	110016
SPW Iterations per vector	1

Results

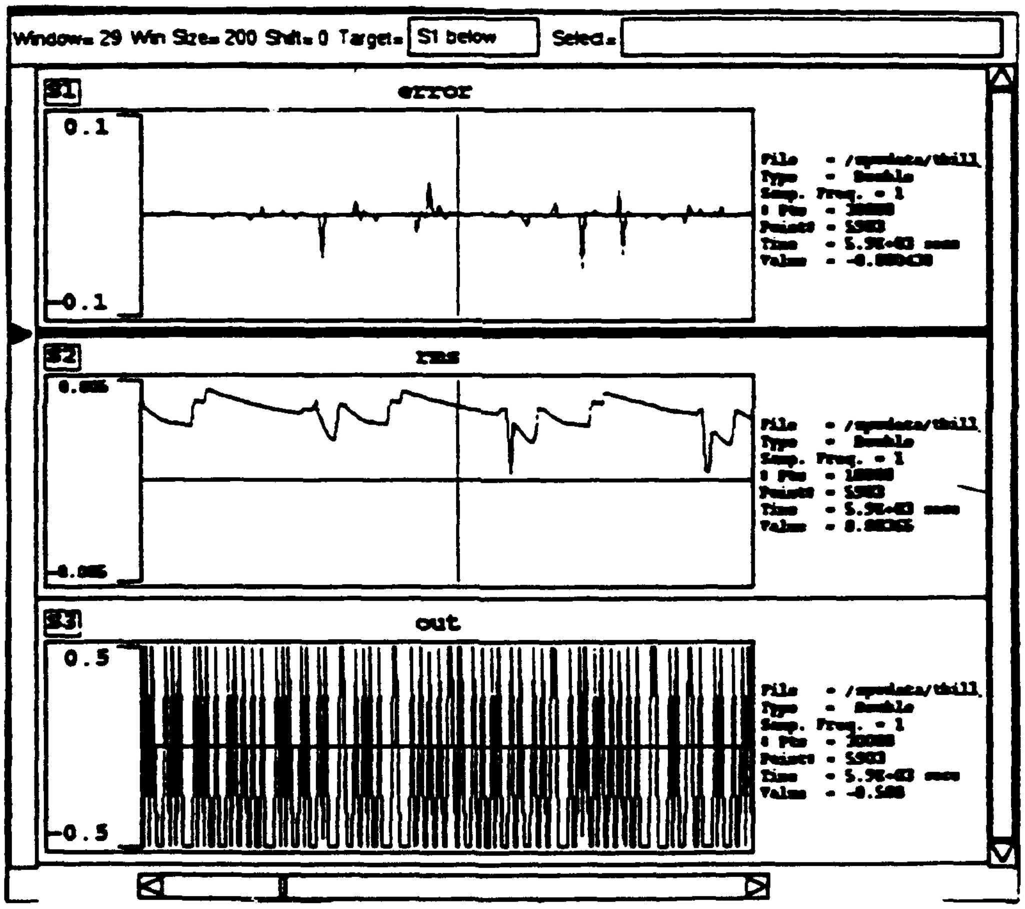
A Backpropagation network with 18 input nodes, 3 hidden nodes, and 3 output nodes (Figure 3) was used to learn the training data set. After 100K iterations, the error, rms error, and output signals appeared as shown in Figure 4.3-1. Note that the error signal at all times was less than about 0.05. Any errors greater than 0.5 would lead to less than 100% accuracy on the training data. This means that the Neural Network output could be thresholded to yield 100% accuracy on the training data.

Potential Extensions

The structure previously described uses a Neural Network to make decisions as to which signal stream is most correct. It does this for multipath cases which the equalizers are previously trained to. This dynamically adaptable equalizer structure can be extended to cover: a) a greater range of multipath delay, b) variation in the relative strength of the primary and secondary signal components, and c) intermediate values of multipath delay and relative strength which fall between the fixed values for which the bank of equalizers have been trained.

Lessons Learned

The Target vector involved in this experiment consisted of values of either -0.5 or +0.5. In other examples it has been observed that training is expedited (when "binary" target values are appropriate) when the target values are not the saturation values for the sigmoid. For examples, -0.4 and +0.4 could be used. However, this particular experiment showed no significant performance advantage using one method over the other.



```
Window= 29 Win Size= 200 Shift= 0 Target= S1 below  Select=

File    = /[illegible]/thill
Type    = [illegible]
Samp. Freq. = 1
# Pts   = [illegible]
Points  = [illegible]
Time    = 5.[illegible] [illegible]
Value   = -0.[illegible]

File    = /[illegible]/thill
Type    = [illegible]
Samp. Freq. = 1
# Pts   = [illegible]
Points  = [illegible]
Time    = 5.[illegible] [illegible]
Value   = 0.[illegible]

File    = /[illegible]/thill
Type    = [illegible]
Samp. Freq. = 1
# Pts   = [illegible]
Points  = [illegible]
Time    = 5.[illegible] [illegible]
Value   = -0.5[illegible]
```

Figure 4.3-1 Various Signals in Backpropagation Network Equalization

4.4 DEMODULATION OF NON-LINEARLY DISTORTED 16-QAM

Application

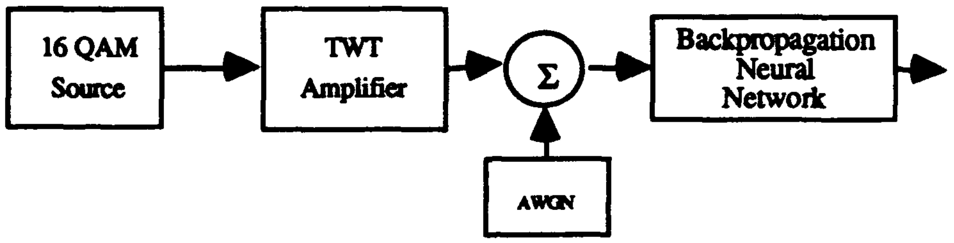
Demodulation of Non-Linearly Distorted 16 QAM using Backpropagation

Introduction

This application examined the use of Neural Network demodulation techniques to counteract the effects of non-linear channels.

Overview

A Backpropagation Neural Network was used to reconstruct the signal constellation of a 16 QAM signal which has been passed through a Travelling Wave Tube amplifier (TWT) and an AWGN channel. The TWT causes the 16 QAM signal to be distorted in a non-linear manner such that the corners of the constellation become rounded.



To restore the transmitted signal constellation, we require a non-linear mapping from the distorted input to the undistorted target. The 64 QAM multipath-distorted signal previously examined (in 4.2 and 4.3) exhibited only linear distortion. In that case, the target symbol was a linear combination of previous and present received symbols. In the case of 16 QAM distorted by a TWT, the Neural Network must emulate a non-linear function.

The training set was 16 received vectors, each of size 2, corresponding to a single sample of each of the 16 QAM symbols. The target vector for each input was the 4-bits (+/- 1) corresponding to each symbol and scaled by 0.4 to remain inside the limits of the output sigmoids.

Traveling Wave Tube Amplifier

This simulation models the AM-to-AM and AM-to-PM characteristics for a typical Traveling Wave Tube (TWT) amplifier. The input and output are complex envelope representations. The equation coefficients and the operating point (dB) are the parameters of the TWT model.

The TWT amplifier is implemented using the following equations for am/am and am/pm conversions [2].

$$A(r) = \frac{a_r r}{1 + B_r r^2}; \qquad \text{Ø}(r) = \frac{a_{\text{Ø}} r^2}{1 + B_{\text{Ø}} r^2}$$

The coefficients a_r, B_r, $a_{\text{Ø}}$, and $b_{\text{Ø}}$ are specified as parameters.

Network Parameters

Paradigm	Backpropagation
Input Nodes	2
Hidden Nodes	16
Output Nodes	4
Learn Rate	0.5, 0.25 *
Momentum	0.5
Update Interval	16
Number of Passes	3125, 3125 *
Training Size	50K, 50K *
SPW Iterations per vector	1

* This network was trained in two phases. The first number gives the value of the parameter in the 1st phase; the second gives the value in the 2nd phase.

Results

We found that the Backpropagation Neural Network was capable of mapping distorted 16 QAM into the corresponding 4-bit vector under varying amounts of noise. Figure 4.4-1 compares the BER for the Neural Network and a "Slicer". The Slicer is simply a decision device whose decision thresholds are fixed to that of ideal 16 QAM. The decision regions formed by the Slicer are shown in Figure 4.4-2.

Lessons Learned

We achieved faster convergene to a solution by modifying the error signal utilized by the Backpropagation Network. Typically, the error signal for a neural network is chosen to be:

e = target - network output

We found that using an error signal equal to

e = target - Quantized network output

provided much faster convergence. We quantized the network output in the following manner:

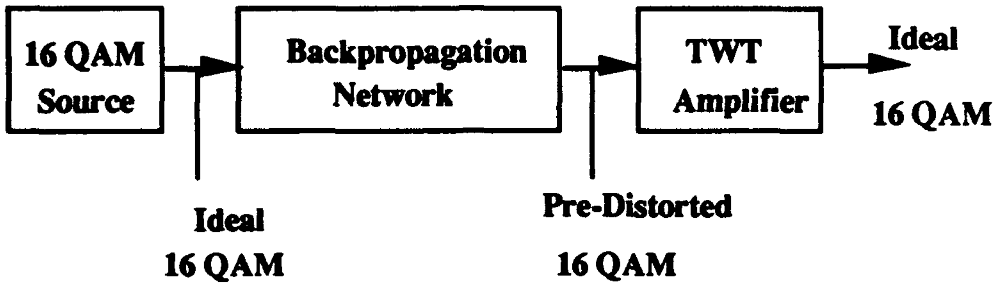
Quantized network output = .4, if the network output > 0

= -.4, if the network output < 0

Note that since each component of the target symbol is either 0.4 or -0.4, this causes the error signal to be 0.8, -0.8, or 0.0. This choice of error signal avoids changing any weights in the neural network when they are producing the correct quantized output.

Potential Extensions

Superior performance could be achieved by pre-distorting a 16 QAM signal prior to transmission in a way that causes TWT amplification to result in the ideal shape instead of distorted QAM.



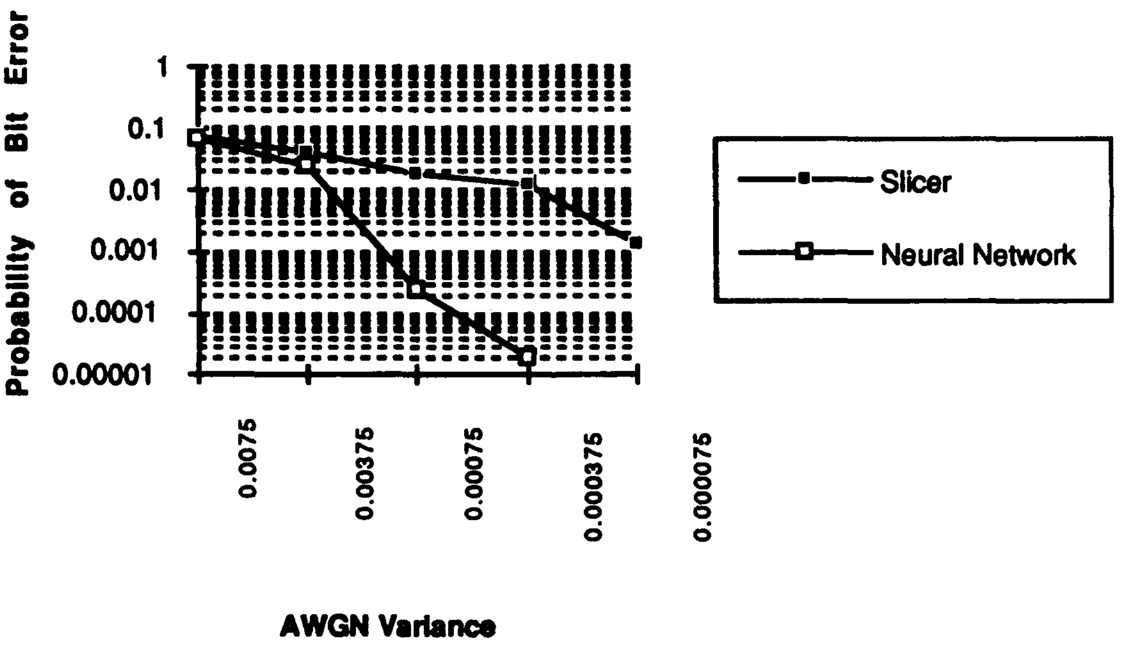
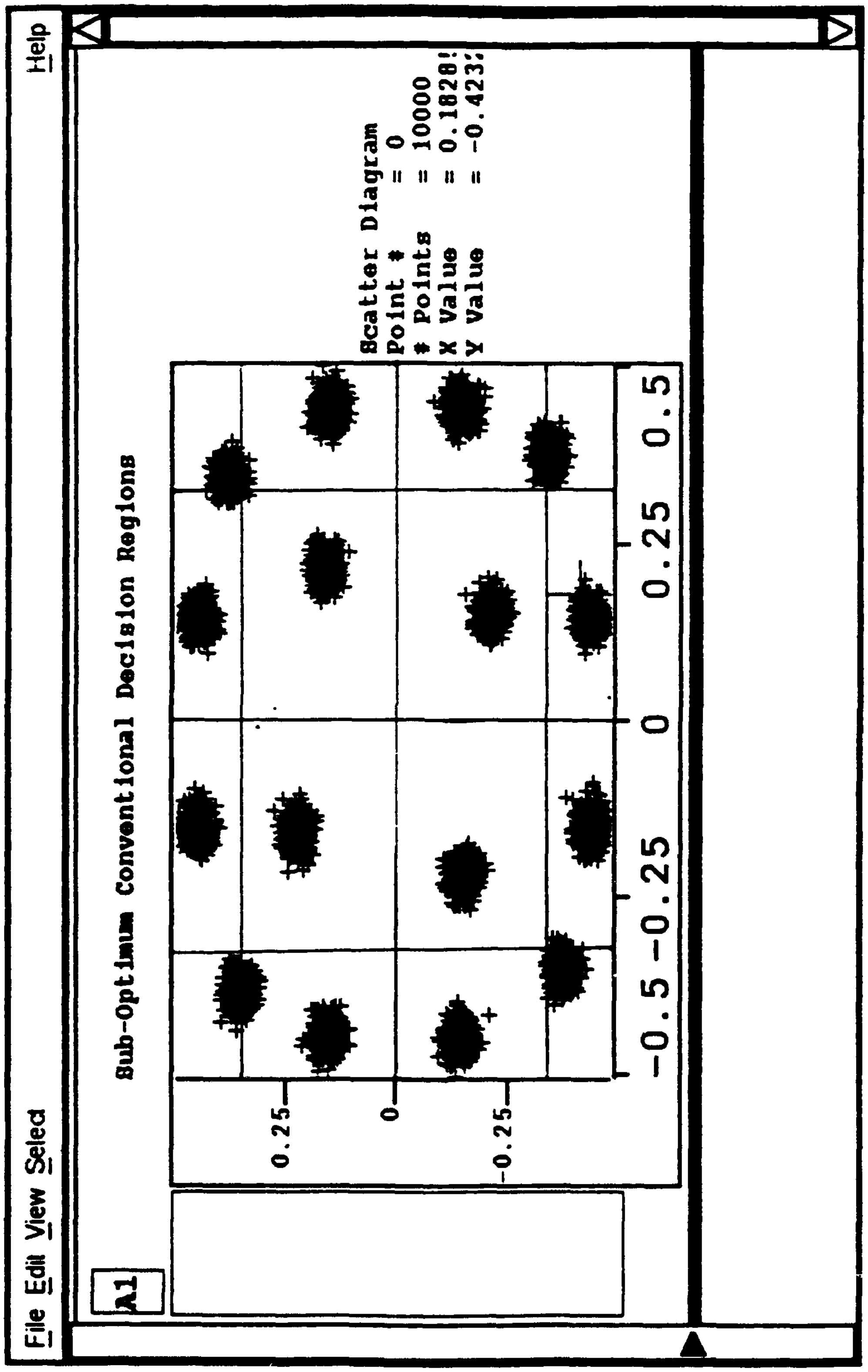


Figure 4.4-1 Bit Error Rate Performance



```
Scatter Diagram
Point #     = 0
# Points    = 10000
X Value     = 0.1828!
Y Value     = -0.423:
```

Figure 4.4-2 Decision Regions for Ideal 16-QAM Compared to Received Constellation

4.5 DEMODULATION OF QPSK WITH BACKPROPAGATION

Application

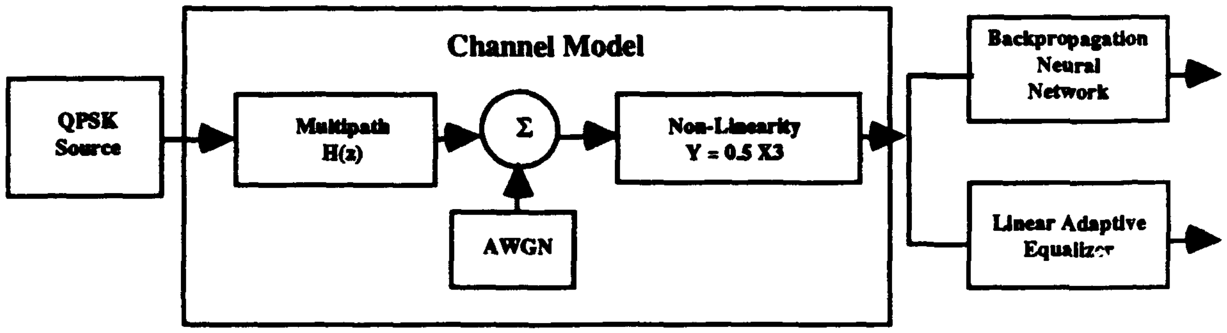
Demodulation of QPSK over a Non-Linear, Dispersive Channel using a Backpropagation Neural Network

Introduction

This application demonstrated a major area of advantage for Neural Networks over conventional techniques. The use of Backpropagation neural networks to adapt to non-linear functions gives considerable improvement over Linear Equalizers.

Overview

QPSK was transmitted over a dispersive, discrete channel using the block diagram below:



The dispersive, discrete channel was modeled by the transfer function :

$$H(z) = 0.3482 + 0.8701z^{-1} + 0.3482z^{-2}$$

where z represents a delay of 1 symbol.

A channel of this form was chosen as in [1] and [2] to emulate a situation where a transmitted symbol interferes with the previous and next symbol due to dispersive effects.

Furthermore, the channel imparts a non-linearity of :

$$y = 0.5x^3$$

where x is the output of the dispersive portion of the channel model and

y is the output of the non-linearity

Finally, AWGN was added.

The I and Q values of the last 3 received symbols was input to the Neural Network, requiring 6 input nodes.

Network Parameters

Paradigm	Backpropagation
Input Nodes	6
Hidden Nodes	8
Output Nodes	2
Learn Rate	0.005
Momentum	0.5
Update Interval	10
Number of Passes	100K
Training Size	1M
SPW Iterations per vector	1

Results

In Figure 4.5-1, Plot A1 shows the signal constellation prior to the non-linearity. Plot A2 shows the constellation at the Equalizer and Neural Network Input. A performance comparison was made between the Linear Adaptive Equalizer and the Backpropagation Neural Network in terms of Bit Error Rate (BER), showing a distinct improvement in favor of the Neural Network (Figure 4.5-2). The Linear Equalizer was unable to compensate for the channel distortion regardless of noise power level while the Neural Equalizer was able to significantly reduce the channel effects and result in a much lower BER.

The BER curves show the results of several methods of applying a Neural Network to this problem:

1. Train the Neural Network to an intermediate or expected value of noise power, then fix the weights.

2. Train the Neural Network on a noiseless case, then fix the weights.

3. Continuously train the Neural Network as the noise power varies.

The BER curves corresponding to the first and second methods described above are shown in the diagram below. A BER curve for the third method would show it to be superior to both of the other methods for all ranges of noise power.

Potential Extensions

Providing for Decision Feedback in the Backpropagation Network should improve performance. Comparisons can then be made to Decision Feedback Equalizers.

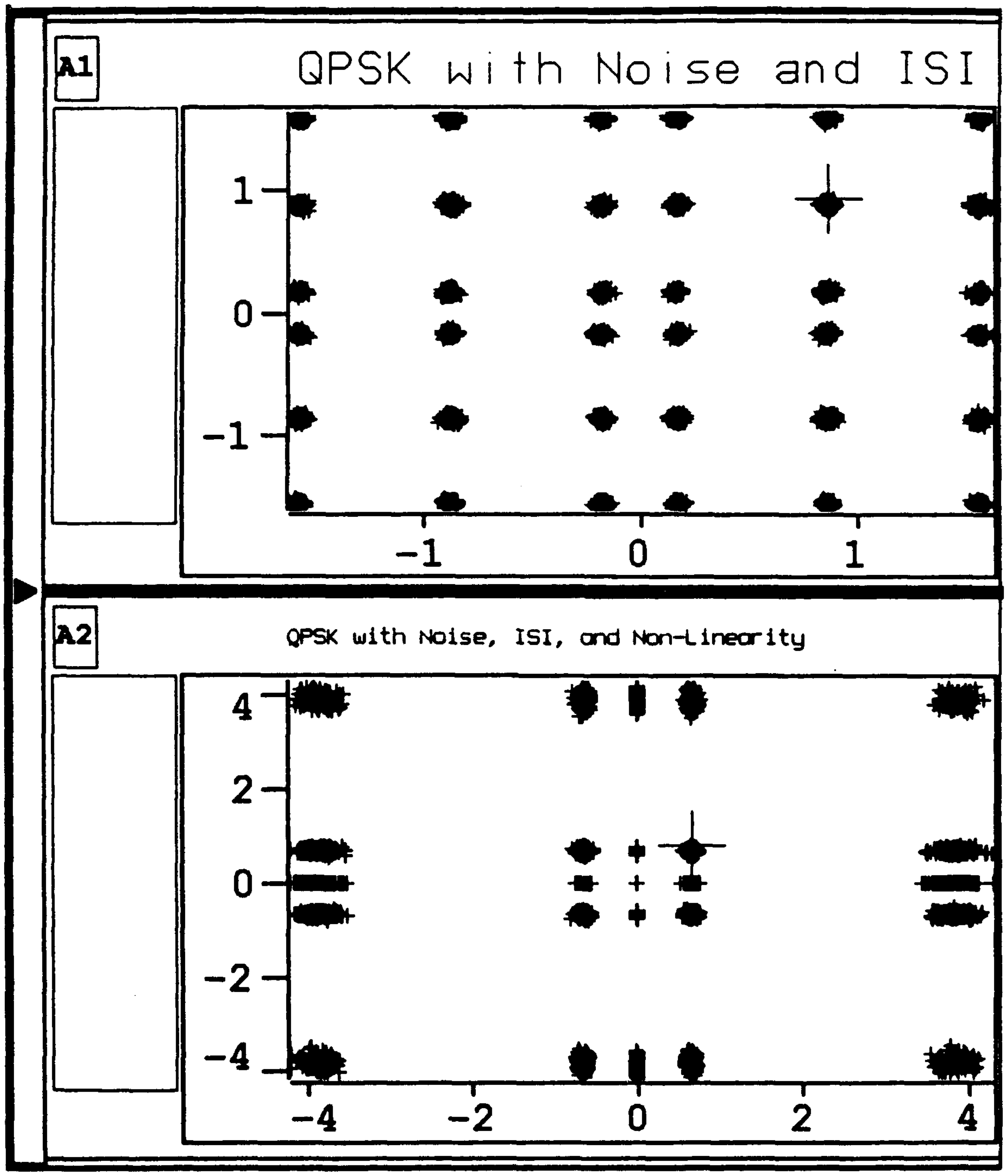


Figure 4.5-1 16-QAM Signal Constellations at Various Points in the Channel Model

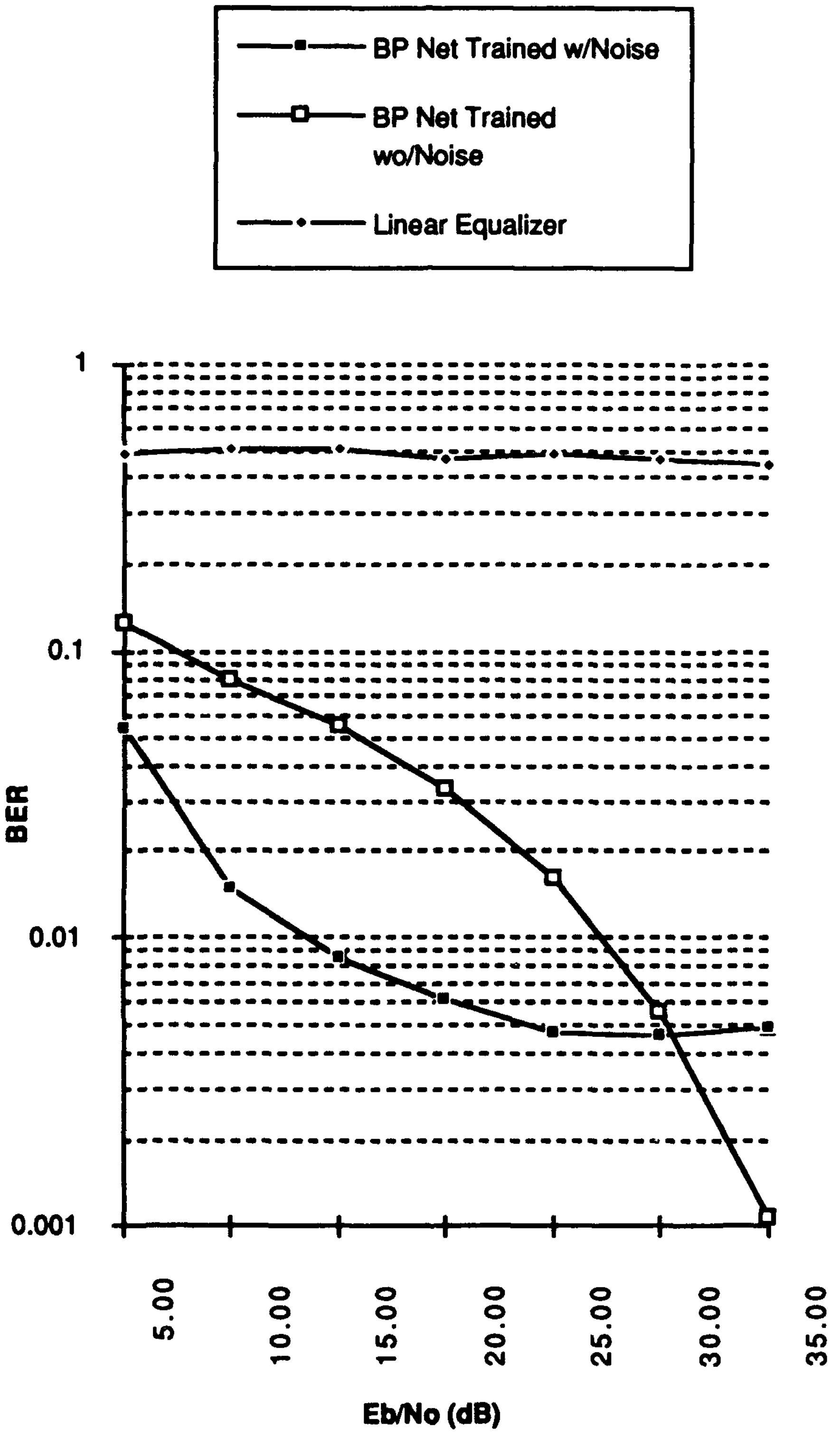


Figure 4.5-2 Bit Error Rate Performance

4.6 DEMODULATION OF QPSK WITH KOHONEN-OUTSTAR

Application

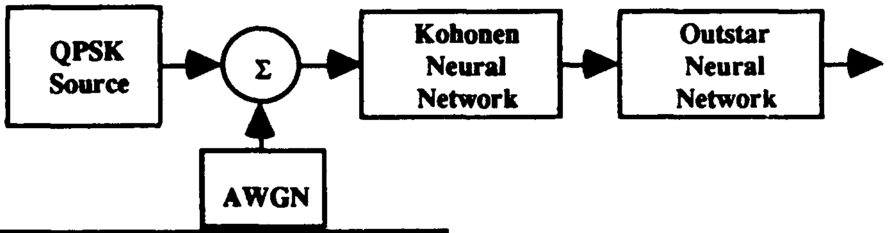
Demodulation of QPSK using a Configuration of Kohonen and Outstar Neural Network

Introduction

The Kohonen Self-Organizing Feature Map is useful in categorizing distributed input vectors into exemplar vectors. Given a received signal constellation, the Kohonen Network was used to adapt its weights without the use of a training signal such that the resulting weights represented the ideal transmitted symbols minus the effects of AWGN.

Overview

Initial examination of Kohonen Topological Map applications to signal processing was done using QPSK transmission over an AWGN Channel. QPSK symbols were represented with complex envelope representation at 1 in-phase and quadrature sample per symbol. QPSK plus noise was input to a Kohonen Neural Network with 16 nodes. The Kohonen output was sent to an Outstar which effectively mapped each winning Kohonen node to a specific QPSK symbol. The trained configuration resulted in a network which could make decisions on received QPSK symbols based upon their Euclidean distance from the ideal QPSK
symbol positions.

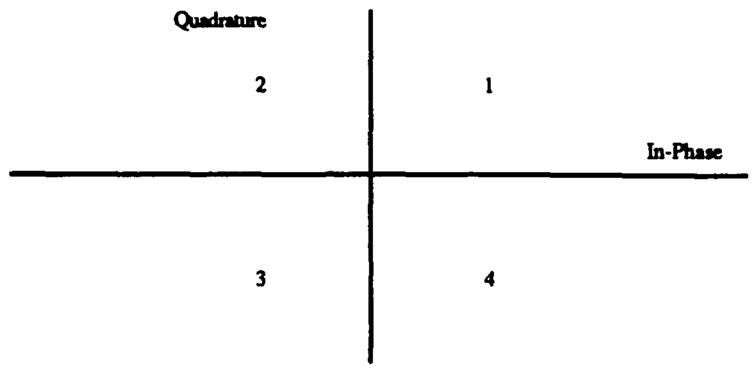


Paradigm	Kohonen	Outstar
Input Nodes	2	16
Hidden Nodes	N/A	N/A
Output Nodes	4	2
Learn Rate	*	0.05
Momentum	N/A	N/A
Update Interval	1	1
Number of Passes	5K	8K
Training Size	20K	20K
SPW Iteration per vector	1	1

Network Parameters

*The Kohonen network was trained in adaptive mode, where weights are changed only for the winning node (neighborhood depth =0). The learning rate was decreased linearly from 1.0 to 0.0 over 5K iterations.

The Kohonen weights will converge to or near the centers of each of the four QPSK clusters in the received signal constellation. The Outstar network simply learns a mapping between a each Kohonen weight and an ideal QPSK symbol. The resulting trained Kohonen/Outstar configuration performs quantization of received QPSK symbols in a manner equivalent to a QPSK slicer.



That is, any received symbol is mapped to the corresponding ideal QPSK symbol based on which quadrant in the signal constellation it lies. For communication constellations with only AWGN distortion, optimum decision regions are determined by linear dissections of the signal constellation. Thus this Neural Network configuration provides no advantage over conventional techniques in terms of performance. However, when there is non-linear distortion present and/or dynamism in the channel, Kohonen/Outstar configurations can give BER advantages due to their ability to form optimum decision regions, under certain conditions. These situations are examined in the next sections.

Results

In Figure 4.6-1, Plot A1 shows the noisy received signal constellation. Plot A2 shows the output constellation over the last 10000 symbols during training. Note that the resulting exemplars have settled to the centroids of the 4 QPSK symbol clusters. Symbol decision for received I-Q pairs may be made based on their distance from each of these exemplars.

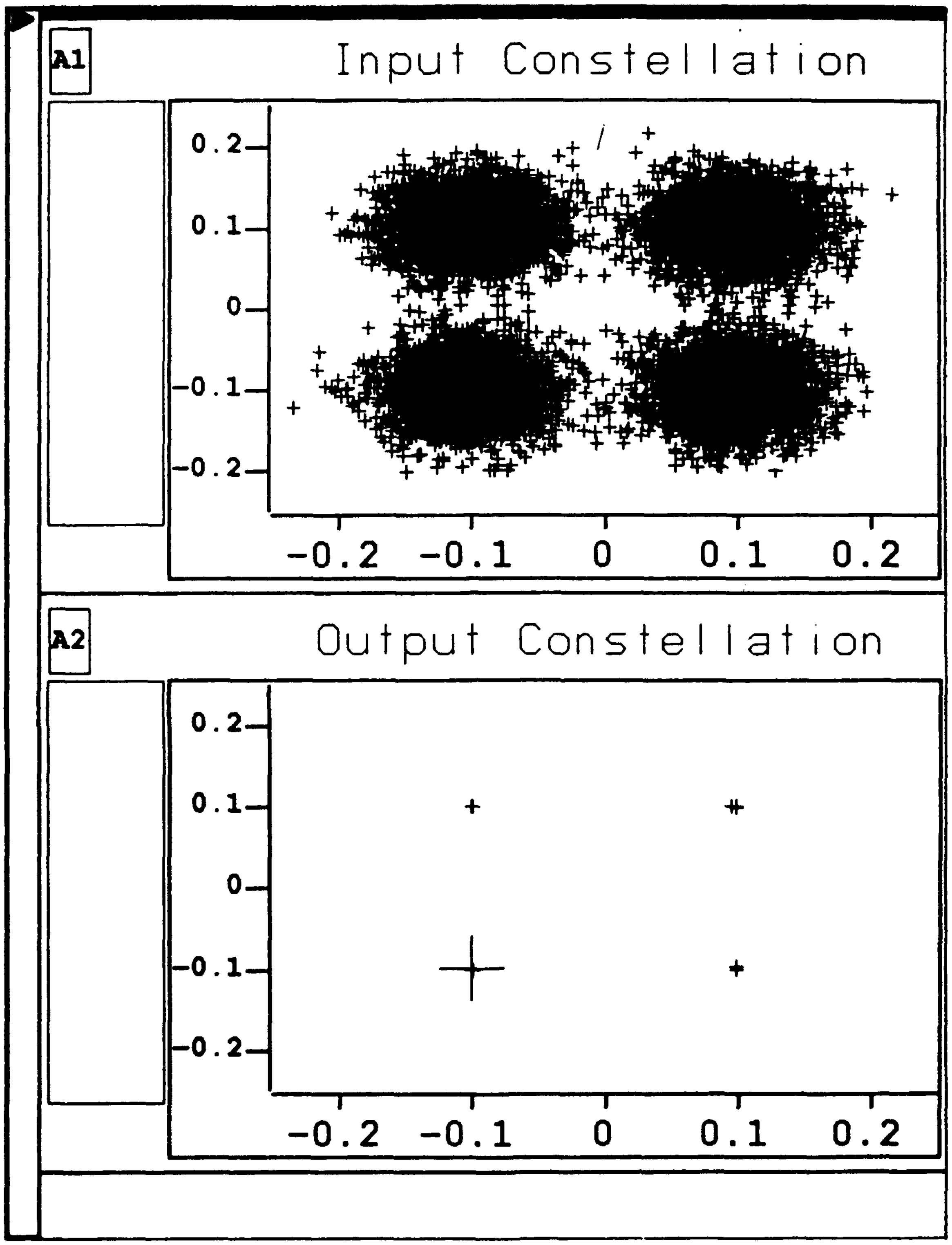


Figure 4.6-1 QPSK Input and Output Constellations

4.7 DEMODULATION OF NON-LINEARLY DISTORTED 16-QAM

Application

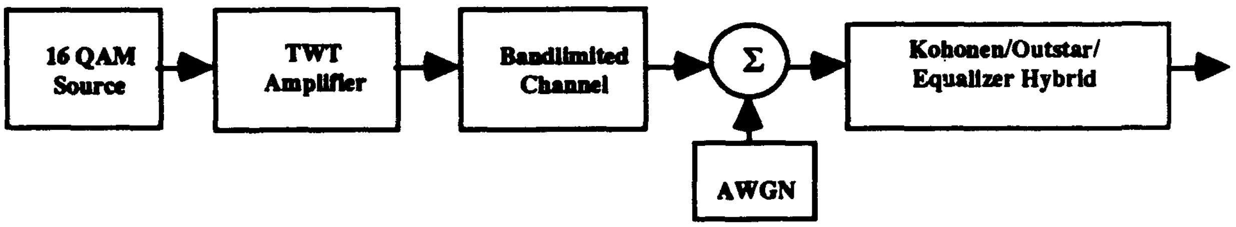
Demodulation of Non-Linearly Distorted 16 QAM using a Configuration of Kohonen and Outstar Neural Networks

Introduction

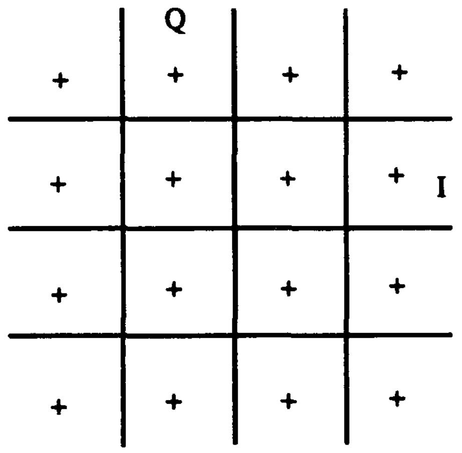
When there is non-linear distortion present and/or dyname variation in the channel, Kohonen/Outstar Neural Network configurations can give improved bit error rate (BER) performance due to their ability to adaptively improve decision regions.

Overview

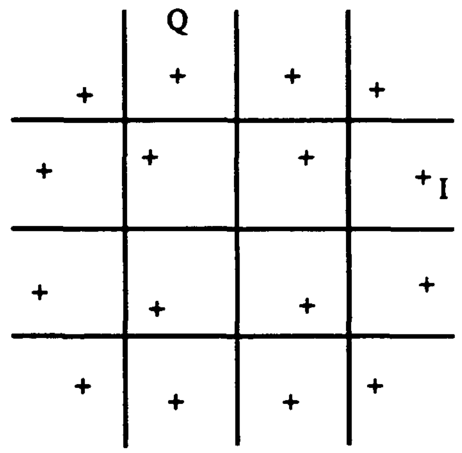
The use of a Kohonen/Outstar configuration can be used to adaptively maintain near-optimum decision regions. A Kohonen network with 16 weights (1 weight for each 16 QAM symbol) will converge to a situation where each weight lies at or near the center of the corresponding distorted QAM symbol cluster, when AWGN is added to the TWT output. Instead of making symbol decisions based upon a linear dissection of the constellation, the Kohonen network will map a received symbol plus noise to the closest (by Euclidean distance) Kohonen weight. The Outstar maps each Kohonen weight to an ideal 16 QAM symbol after a brief training period given a training signal. In this manner, BER performance is improved with the neural configuration.



The Travelling Wave Tube (TWT) Amplifier produces a compression of the 16 QAM signal constellation due to AM-AM and AM-PM conversion. The non-linear effects worsen as the output power increases. For this reason, output power levels beyond a certain point are not feasible using conventional techniques.



Decision Regions are Optimum

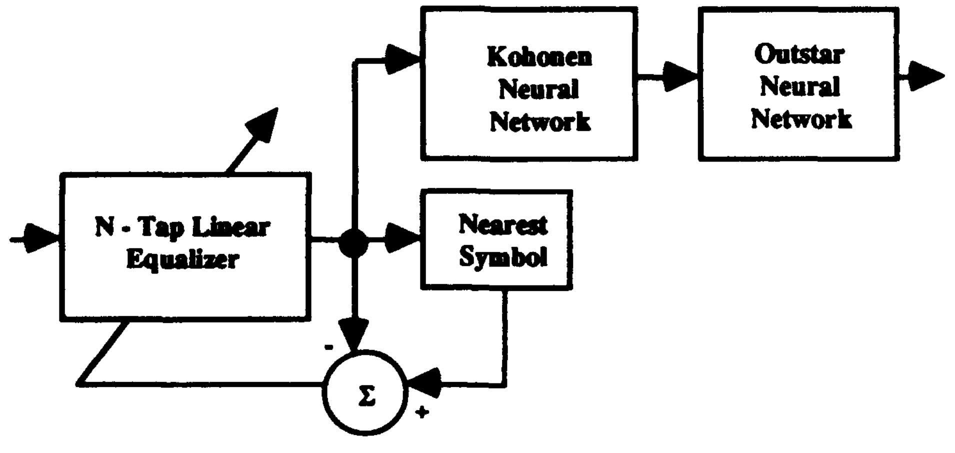


Decision Regions are Sub-Optimum

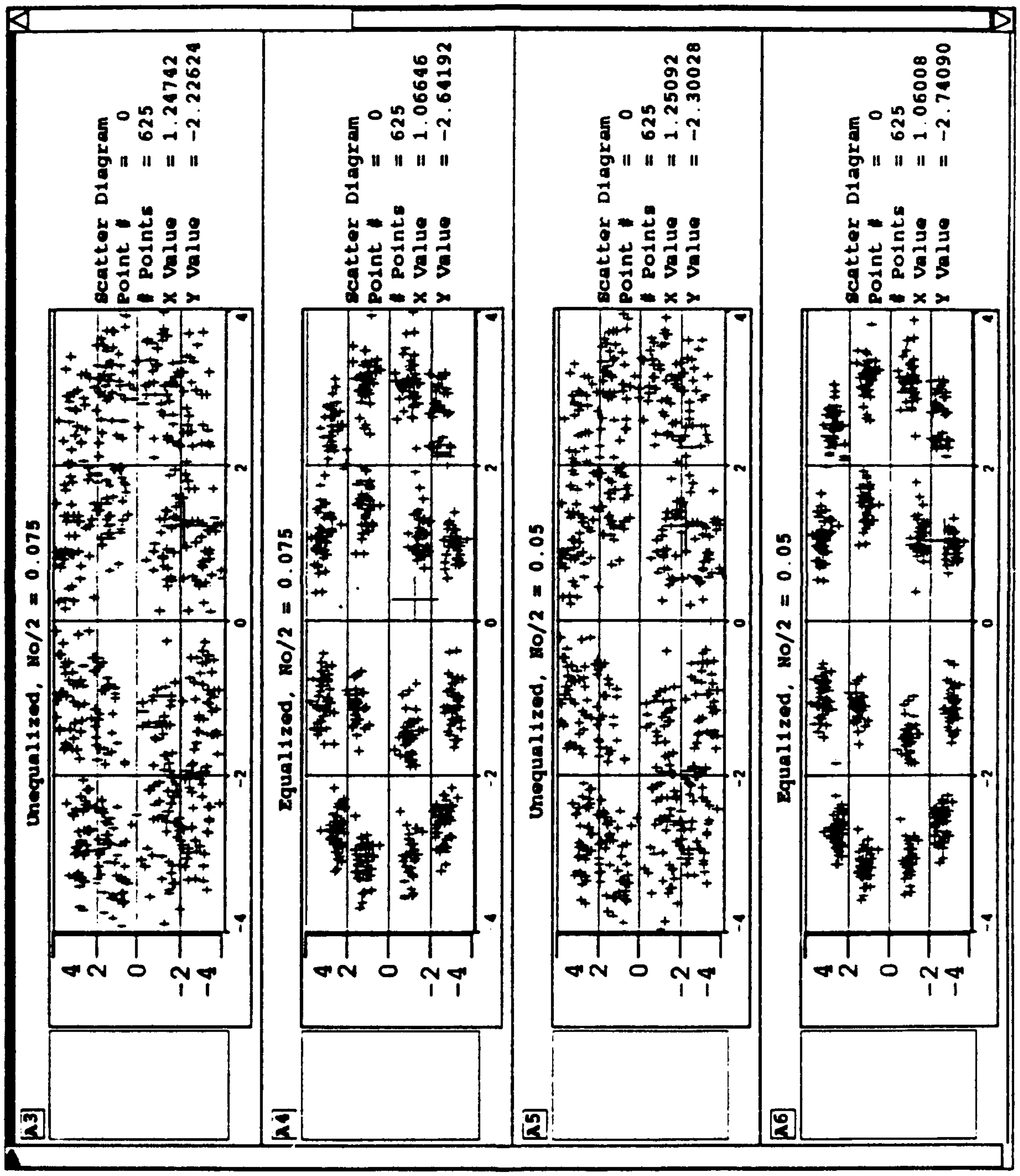
The constellation diagram in Figure 4.7-1, Plots A3 and A5 display the resulting signal for 16 QAM through a TWT, through a bandlimited channel, plus AWGN.

The Kohonen network will not converge to the centers of the transmitted symbols. However, if the effects of ISI were eliminated, the resulting constellation would be as in Figure 4.7-1, Plots A4 and A6, and the Kohonen network could converge to cluster centers. The removal of ISI is easily performed by linear equalization. A Backpropagation Neural Network has also been shown to be capable of linear equalization.

The following structure is capable of demodulating a received signal as in Figure 4.7-1, Plots A4 and A6:



In this structure, an N-tap Linear Equalizer is used to remove the ISI. This may be done conventionally as shown, or with a Backpropagation Network. In the diagram above, the output of the equalizer is quantized to the nearest symbol. The difference between the quantized and unquantized output is an error signal which is scaled and fed back to adjust the weights of the equalizer in an adaptive manner. In the structure above, symbol decisions are made based on the nearest Kohonen weight vector. Given that the ISI is removed by the equalizer, the Kohonen Network can be used to find the centers of the resulting symbol clusters, which represent the optimum or near-optimum quantal levels for making symbol decisions. An Outstar Network maps each Kohonen weight vector to an ideal symbol value. Thus, the Kohonen/Outstar configuration in tandem with the Linear Equalizer (or Backpropagation Network) yields better performance than either alone.



```
Scatter Diagram
Point #    = 0
# Points   = 625
X Value    = 1.24742
Y Value    = -2.22624

Scatter Diagram
Point #    = 0
# Points   = 625
X Value    = 1.06646
Y Value    = -2.64192

Scatter Diagram
Point #    = 0
# Points   = 625
X Value    = 1.25092
Y Value    = -2.30028

Scatter Diagram
Point #    = 0
# Points   = 625
X Value    = 1.06008
Y Value    = -2.74090
```

Figure 4.7-1 Unequalized and Equalized 16-QAM Signal Constellations for Varying Degrees of Channel Noise

Network Parameters

Paradigm	Kohonen	Outstar
Input Nodes	2	16
Hidden Nodes	N/A	N/A
Output Nodes	16	2
Learn Rate	*	0.05
Momentum	N/A	0.05
Update Interval	1	1
Number of Passes	*	25K
Training Size	*	25K
SPW Iteration per vector	16	16

* Kohonen network was trained in two phases. In Phase 1, training proceded conventionally with a learning rate of 0.7. Phase 1 training was in effect for 16K iterations. At the completion of Phase 1 training, Phase 2 training began. Phase 2 training was in adaptive mode where only the weights to the winning Kohonen node were updated. The learning rate during Phase 2 training was 0.1.

Results

Figure 4.7-2 compares the BER of a Linear Equalizer alone against that of the Kohonen/Outstar/Equalizer hybrid.

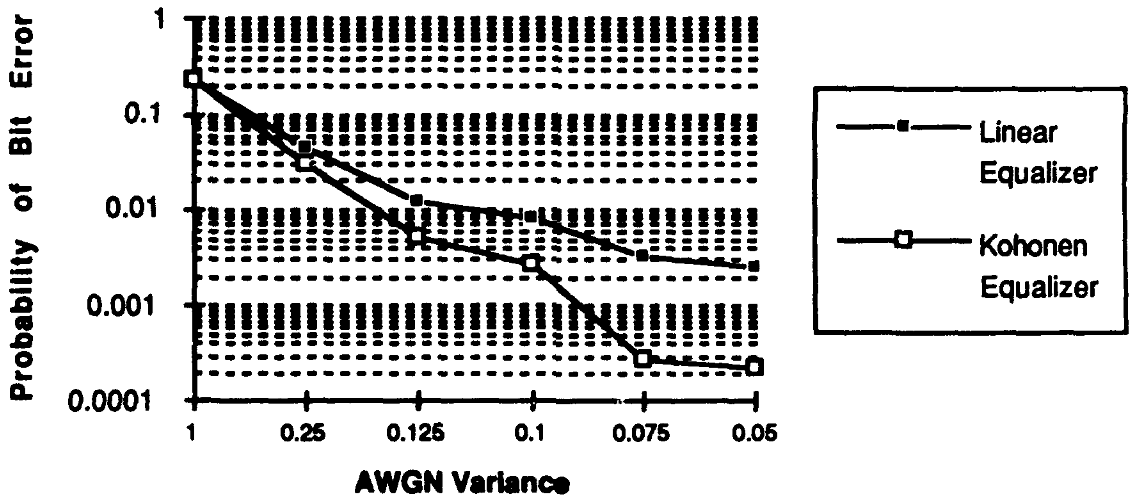


Figure 4.7-2 Bit Error Rate Performance

4.8 DEMODULATION OF 16-QAM OVER A RAYLEIGH CHANNEL

Application

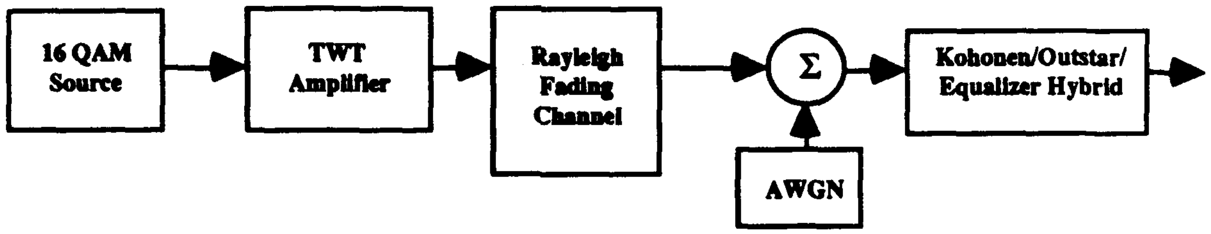
Demodulation of 16 QAM over a Rayleigh Channel using a Configuration of Kohonen and Outstar Neural Networks

Introduction

We investigated the use of a combined Linear Equalizer and Kohonen network to "learn and track" the dynamic movement of a 16 QAM constellation when subjected to a Rayleigh Fading Channel.

Overview

A more extensive application of the Kohonen network has been formulated. Mobile communication experiences dynamic channel distortion which has been modeled by Rayleigh Fading Channels. In such channels, the received signal constellation is both rotated and attenuated as functions of time. Typically, PSK-based communication schemes are used in mobile communications since all transmitted symbols are of the same magnitude. This allows for Automatic Gain Control to compensate for the variable attenuation induced by the channel. However, PSK-based communication is not as spectrally efficient as QAM-based communication. Use of Automatic Gain Control for QAM is more complex because QAM symbols are of different magnitudes. The application of the Equalizer/Kohonen hybrid to the Rayleigh channel was examined using the block diagram below:



Rayleigh Fading Channel

The Rayleigh channel in the above block diagram represents the flat (or single ray) Rayleigh fading channel model used to model the effects of multiple point scatters i the neighborhood of a moving receiver in mobile communications.

The output of the block is simply the input times a single complex time-varying weight. The weight is called the Rayleigh channel weight and is generated by passing complex white Gaussian noise through a fading filter and then interpolated the output of the fading filter. The fading filter, also referred to as the spectrum shaping filter, is based on Jake's model [Jake, 1974] and has a frequency response of,

$$H(f) = \frac{1}{[1 - (f/f_d)^2]^{0.5}}, \quad \text{if } |f| \leq f_d$$

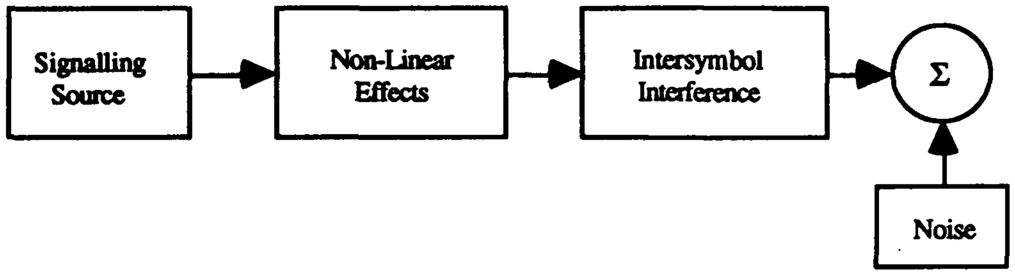
$$H(f) = 0, \quad \text{if } |f| > f_d$$

where f_d is the Doppler frequency.

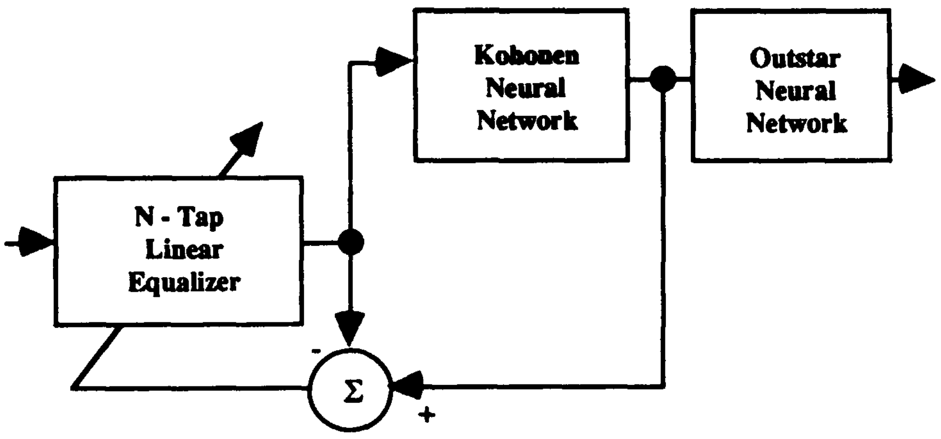
Since the Doppler frequency is usually much less than the sampling frequency the fading filter response H(f) is usually a very narrow lowpass filter.

Kohonen Equalizer

In an earlier experiment, we found that a Kohonen/Outstar configuration could be used to form near-optimum decision regions for a transmitted signal constellation that has been corrupted by non-linear effects, ISI, and noise.



The final extension to the above configuration is useful for signaling which has been distorted by non-linearity, ISI, and noise in a ***dynamic*** sense. Consider using the winning Kohonen weight vector as the target signal for the computation of the feedback error signal. If the received constellation of Figure 4.7-1 Plot A4 is also rotating, compressing, and expanding, as in a Rayleigh channel, the conventional equalizer will not be able to keep up under certain conditions. A modification to the Kohonen learning algorithm allows the weight vectors to track the movement of the constellation, thus relaxing the rate at which the linear equalizer must adapt. The use of the winning Kohonen weight vector for the computation of the error signal further assists the adaptation. It is also hypothesized that a conventional equalizer will be unable to eliminate ISI if the transmitted constellation is non-linearly distorted beyond a certain point ***unless*** given a more accurate desired (or target) signal from which to derive the error signal. The configuration below can accommodate this more accurate target signal via feedback from the winning Kohonen weight vector.



Network Parameters

Paradigm	Kohonen	Outstar
Input Nodes	2	16
Hidden Nodes	N/A	n/A
Output Nodes	16	2
Learn Rate	0.7	0.05
Momentum	N/A	0.05
Update Interval	1	1
Number of Passes	*	12K
Training Size	*	12K
SPW Iteration per vector	1	1

* Kohonen network was trained in two phases. In Phase 1, training proceded conventionally with a learning rate of 0.7. Phase 1 training was in effect for 16K iterations. At the completion of Phase 1 training, Phase 2 training began. Phase 2 training was in adaptive mode where only the weights to the winning Kohonen node were updated. The learning rate during Phase 2 training was1.0

Results

Results of this experiment are best displayed via the corresponding interactive SPW demonstration. Refer to the Appendix for the demonstration procedures.

4.9 IMPROVING SOFT DECISIONS IN A JAMMING ENVIRONMENT

Application

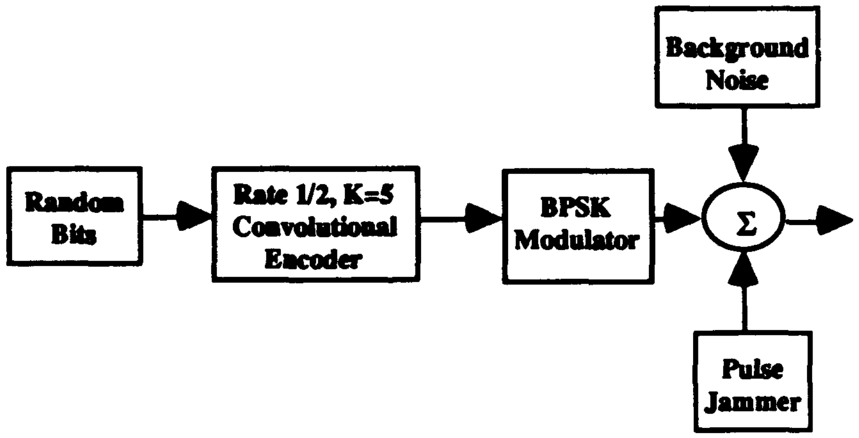
Improving Viterbi Decoder Soft Decisions in a Pulse Jamming Environment

Introduction

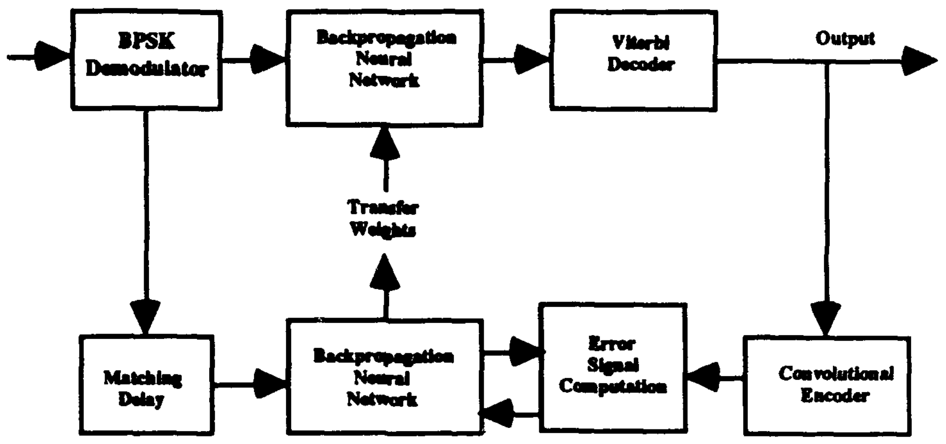
Convolutional coding of data has been shown to improve performance of communications systems in the presence of additive white Gaussian noise (AWGN). Decoding of convolutionally encoded data in an AWGN scenario is optimally done via Viterbi decoding. Supplying soft decisions to a Viterbi decoder instead of hard decisions can give a performance increase of approximately 2 dB. We propose an adaptive, non-linear method of supplying the soft decisions which yields a performance gain in a pulse jamming environment. This technique is similar to that applied in several papers [Asato, Grover & Cahn, "Artificial Neural Network Adaptive Non-Linear Digital Receivers"], [Anderson, "Generation of Soft Information in a Frequency Hopping HF Radio System Using Neural Networks" Milcon '92 Proceedings Vol. 2]. This adaptive receiver structure can adjust to varying jammer conditions. It requires no training signal, no knowledge of pulse jammer duty cycle, channel error rate, or jammer magnitude. If the jammer were to be permanently turned off or its duty cycle were to change, the network would change its soft decision metric function appropriately.

Overview

This neural network application was studied using the block diagrams below:



Transmitter and Channel Model



Receiver Model

Random binary data with equally likely probability of 0 and 1 was encoded by a rate 1/2, constraint length 5 convolutional encoder. This encoded stream modulated a binary phase -shift keying (BPSK) tramsitter, whose outputs were +1 or -1. The channel consisted of background AWGN and a pulse jammer as described below. The received signal was demodulated and input to a Backpropagation Neural network which adaptively generates a soft decision metric for input to the Viterbi decoder. The Viterbi decoder utilizes the soft decisions quantized to 8 levels to produce the estimate of the transmitted data stream.

The weights of a Backpropagation Neural Network are adjusted via an error signal which is calculated based upon a comparison between the network output and a target signal. In some applications, a training signal is available. In situations where a training signal is not available, the target signal must be estimated from information at hand. In this application, a fairly good estimate of the transmitted signal is the Viterbi decoder output. The neural network will produce a soft decision metric which will be used by the decoder to produce an estimate of the message data. The job of the neural network is to learn to use the demodulator output to give the decoder an accurate metric. For this reason, we re-encode the decoder output for use in calculating the error signal. The re-encoded decoder output is a good estimate of the actual signal for comparison with the received noisy signal.

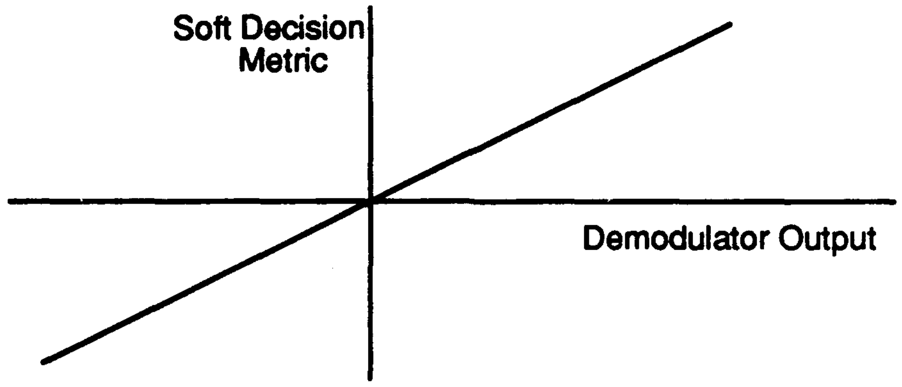
The Viterbi decoder and encoder contain internal delays which must be accounted for when using their output in calculation of error signals. For this reason, two Backpropagation Neural Networks are used. The error signal is applied to the bottom neural network. The input to the bottom neural network is a delayed version of the demodulator output, in order to compensate for internal decoder and encoder delays. The weights of the bottom neural network are transferred to the top neural network immediately upon change for use in calculating the soft decision metrics.

Ideally, the soft decision metric given to the Viterbi decoder for an arbitrary demodulated signal level is the log-likelihood function. It is the logarithm of the likelihood ratio which is the quotient of the probability of bit correctness divided by the probability of bit error, given the demodulated signal:

$$\text{Metric} = \log \frac{\Pr(\text{correct/demodulated value})}{\Pr(\text{incorrect/demodulated value})}$$

The Viterbi decoding algorithm for convolutional codes is equivalent to maximum likelihood decoding and thus is optimum for equally likely messages.

Most conventional soft decision implementations are designed for stationary AWGN degradation. In many situations, channel noise is not stationary, such as in pulse jamming environments. In AWGN environments, the ideal soft decision function is simply a linear function of the demodulator output. For BPSK signalling with transmitted symbols +1 and -1, demodulator outputs near 0 would be assigned a small metric since it is not certain whether a +1 or -1 was actually transmitted. Large positive (or negative) demodulator outputs would be assigned a large positive(or negative) metric since it is nearly certain that a +1 (or -1) was transmitted since AWGN is unlikely to account for such a change in received signal voltage.



Conventional Soft Decision Metric

Pulse Jammer

The pulse jammer is modeled as an AWGN noise source that is switched on and off at a particular duty cycle. When the jammer is off, the noise caused by the channel is simply that of the background noise, No/2. When the jammer is on, the noise power is Nj/2 >> No/2. This jammer will cause bits transmitted during the "on" times to vary greatly in magnitude. Thus it is much less certain whether a +1 or -1 was transmitted. For this reason, that bit should be assigned a low metric.

Error Signal

The error signal, E, is the difference between the Neural Network Target and the Neural Network Output (OUT)

$$E = \text{Target} - \text{OUT}$$

The Target, and hence E, is dependent upon a comparison between the quanitized demodulated signal and the encoder output.

Let D denote the output of the BPSK demodulator, and define the "sign" of D to be:

S(D) = +1 if D is positive or zero, and

S(D) = -1 if D is negative.

Also define the demodulator's "hard" bit decision to be:

H(D) = 1 if D is positive or zero, and

H(D) = 0 if D is negative.

Then the Target is defined to be:

$\text{Target} = S(D)\lvert OUT\rvert(1+{}^{1}/_{R})$	if $H(D)$ = encoder output
$\text{Target} = 0$	if $H(D)$ does not equal encoder output

In this equation, R denotes the estimated likelihood ratio.

Since the Neural Network Output, OUT, is trained to the log-likelihood function, the estimated likelihood ratio, R, is defined as:

$$R = 10^{\lvert OUT\rvert}$$

Thus when the neural network has supplied a metric which has resulted in an incorrect bit decision, the metric is driven towards 0. When a correct bit decision is made, the metric is re-inforced by an amount $(1 + 1/R)$ which ideally will balance the metric at a value according to its likelihood ratio. For example, if a given demodulated value D has a likelihood ratio of R_D, then $1/R_D$ of the times that D is input to the network will result in an incorrect bit decision, giving a Target of 0. To offset this and allow the network to stabilize at the true value of $\log(R_D)$, we must supply a target of $1+1/R$ times the network output when a correct bit comparison is made.

Simulation Parameters

Paradigm	Backpropagation
Input Nodes	1
Hidden Nodes	10
Output Nodes	1
Learn Rate	*
Momentum	0.0
Update Interval	1
Number of Passes	Always in training
Training Size	N/A
SPW Iteration per vector	1

* Linearly decreasing from 9.0 at the rate of -4.5t/40K (where t is the SPW iteration count) over the 1st 25K iterations. Learning rate equals 0.1 afterwards.

The neural network was initally trained
to an identity function to approximate the conventional soft decision metric. These initial conditions are necessary for the network to give reasonable metrics to the Viteribi decoder at startup for convergence to occur. Training consisted of varying the learning rate from 9.0 to 4.5 linearly over the first 25K iterations, then applying a constant 0.1 learning rate

over the next 40K iterations. The initial large learning rates accelerated the learning process to a point where finer adjustment with a learning rate of 0.1 could begin.

Theoretical Background

The benefit of the neural net approach is to enable the use of near-optimal log-likelihood ratios without requiring any knowledge of jammer power or duty factor. It automatically learns this during operation, and it tracks any changes in these characteristics as the jamming environment changes. To see how important this might be we can calculate the ideal log-likelihood ratio that would be used as a metric and examine the changes in the ratio as the jamming environment changes. The demodulator output probability density function as a function of the output voltage n is the Gaussian pdf, N(n,m,s), where

$$N(\nu,\mu,\sigma)=\frac{e^{-(\nu-\mu)^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}}$$

and where the demodulator output voltage in the absence of noise is assumed to be m and the standard deviation of the noise process is s. The correctly received signal is assumed to have m = $\sqrt{E_s}$ where E_S is the received energy per channel symbol. The noise processes are assumed to be such that when the pulse jammer is on a fraction d of the time the noise standard deviation is s = $\sqrt{N_J/2}$, and the other fraction (1-d) of the time the noise standard deviation is s = $\sqrt{N_o/2}$, i.e., that of the background noise. Thus, the equation for the ideal log-likelihood ratio as a function of the demodulator output voltage, n, the signal mean, m = $\sqrt{E_s}$, the jammer duty factor, d, and the background and jammer noise standard deviations, $\sqrt{N_o/2}$ and $\sqrt{N_J/2}$ is given by LLR(n) where

$$\mathrm{LLR}(n)=\log\frac{(1-d)\,N(n,\sqrt{E_s},\sqrt{N_o/2})+d\,N(n,\sqrt{E_s},\sqrt{N_J/2})}{(1-d)\,N(n,-\sqrt{E_s},\sqrt{N_o/2})+d\,N(n,-\sqrt{E_s},\sqrt{N_J/2})}\qquad(2)$$

Behavior of the ideal log-likelihood ratio as given by (2) is shown in Fig. 1 for a pulse jammer 20 dB larger than the background noise and with duty factors of 0.05, 0.1, and 0.15. The log-likelihood ratio departs significantly from the ideal linear case for only background noise, but there is not a great deal of variation as the jammer duty factor, s is varied. The main effect is that the positive and negative peaks are reduced as the duty factor increases because the reliability of decisions at those voltages decreases. In contrast, there is a much more significant change in the shape of the ideal log-likelihood function as jammer power varies with constant jammer duty factor as shown in Fig. 2. The main effect is that as the jammer power is reduced, the reliability of bit decisions for larger net input voltages is improved significantly causing the log-likelihood ratio to increase. Hopefully, the neural net can converge to a log-likelihood function that is quite close to the optimal. In this process it is most important that the neural net provide near zero log-likelihoods for input voltages corresponding to unreliable symbols. This allows the Viterbi decoder to treat such symbols as unreliable in accumulating path metrics thereby minimizing their effect on decoder bit decisions. Performance will be significantly degraded if larger log-likelihood ratios are produced for input voltages corresponding to a high percentage of unreliable symbols. What this shows is that there is a need to have a neural net approach that can adaptively track changing jamming conditions to provide the decoder with the best log-likelihood ratio metrics at a given time. As part of our development plan we want to insure that the approach does the best possible job of converging to nearoptimal metrics while having the capability to quickly track changes in noise conditions.

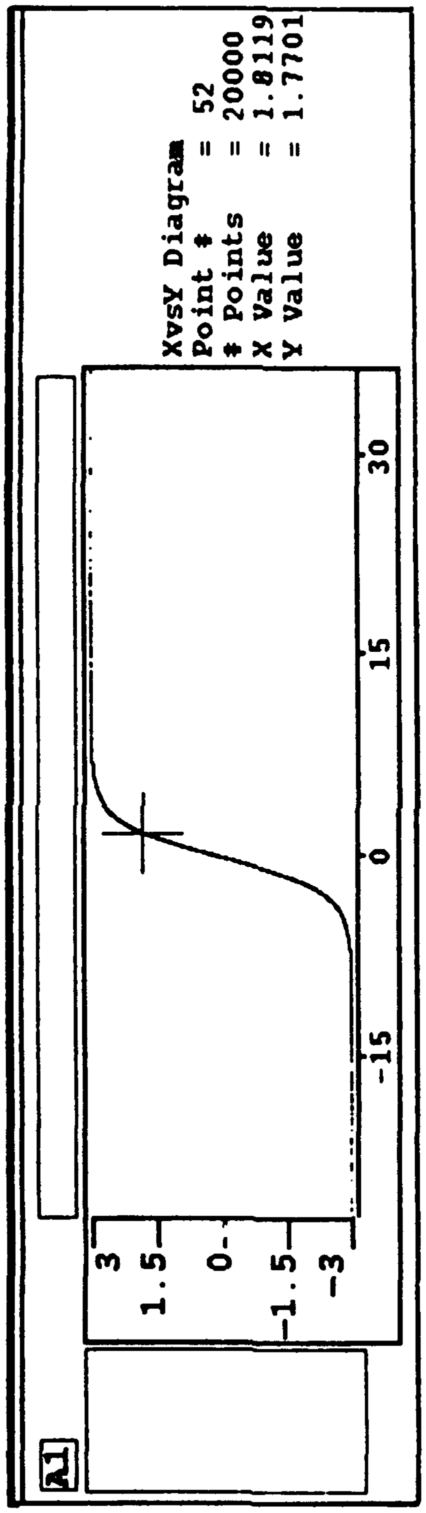
Results

Figures 4.9-1 and 4.9-2 displays the soft decision metric transfer function produced by the network before, during, and after training for a 5% jammer which is 20dB greater in power than the background signal-to-noise ratio. E_b/N_o is 4.5 dB.

Figure 4.9-1 is the initial transfer function, with a linear characteristic which is optimal for an AWGN environment. Figure 4.9-2 Plot A1 displays the transfer function near the end of training. Plot A2 shows the final transfer function.

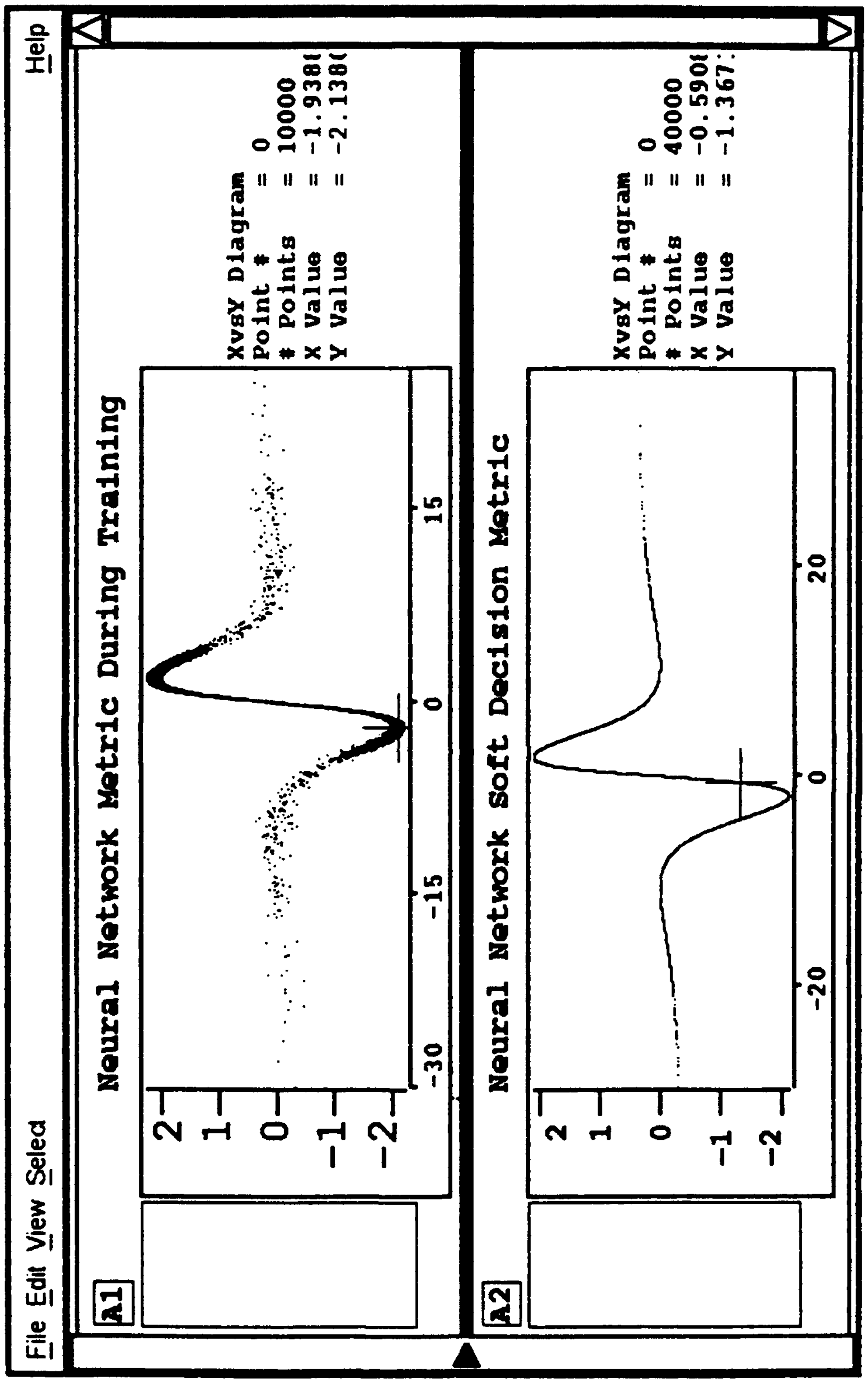
Figure 4.9-3 is a plot of the upper half of the theoretical optimum transfer function.

Figure 4.9-4 is a graph of Bit Error Rate curves comparing the neural network performance to a conventional linear soft decision metric and theoretical bounds. The theoretical curves assume infinite soft decision quantization, knowledge of pulse jammer on/off times, and knowledge of the relative magnitude of the jammer over background noise.



```
XvsY Diagram
Point #     = 52
# Points    = 20000
X Value     = 1.8119
Y Value     = 1.7701
```

Figure 4.9-1 Initial Neural Network Soft-Decision Metric (Before Adaptation)


File Edit View Select
Help
A1
Neural Network Metric During Training
2
1
0
-1
-2
-30
-15
0
15
A2
Neural Network Soft Decision Metric
2
1
0
-1
-2
-20
0
20

```
XvsY Diagram
Point #    = 0
# Points   = 10000
X Value    = -1.938
Y Value    = -2.138
```

```
XvsY Diagram
Point #    = 0
# Points   = 40000
X Value    = -0.590
Y Value    = -1.367
```

Figure 4.9-2   Neural Network Soft-Decision Metric During and After Adaptation

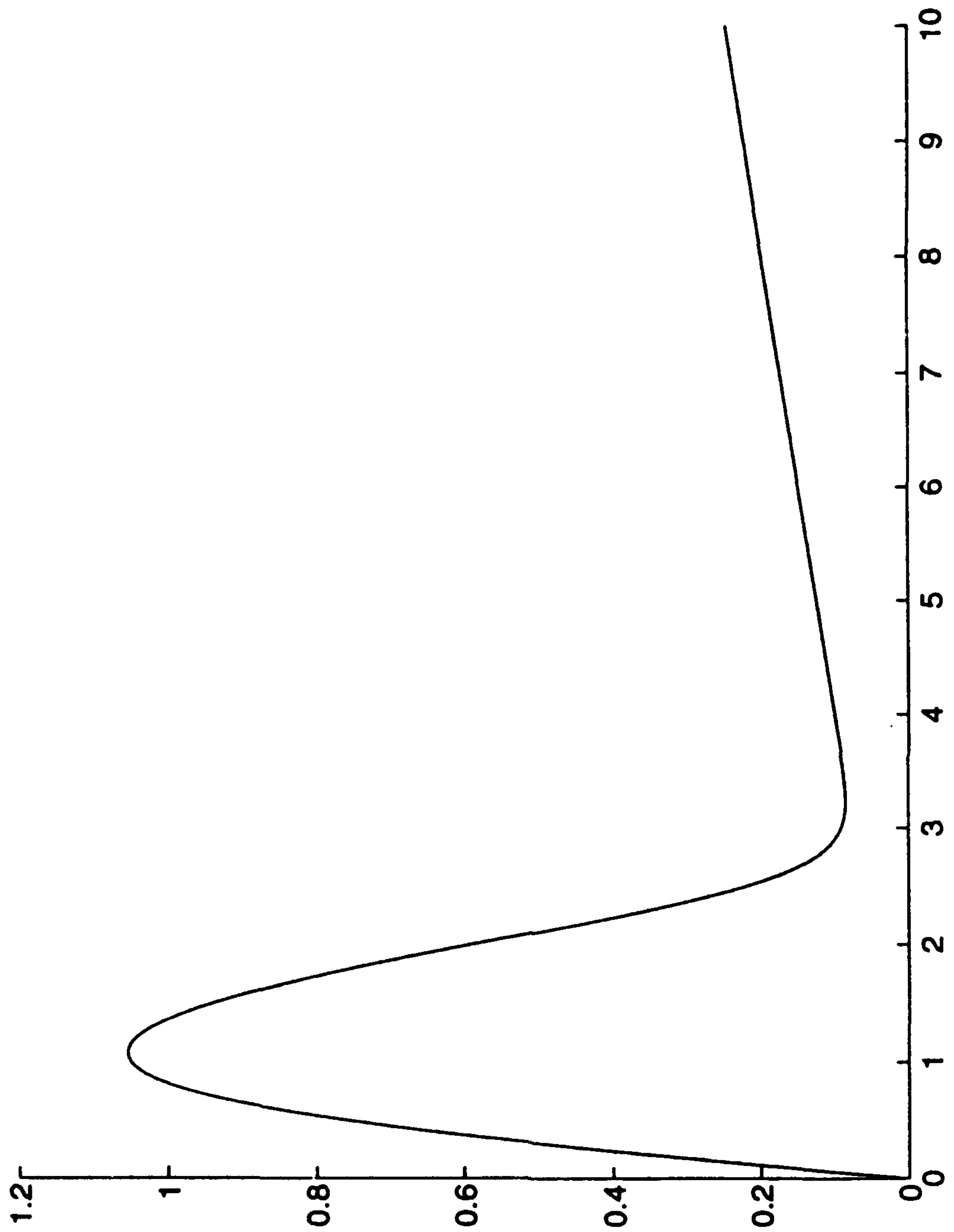


Figure 4.9-3 Optimum Soft-Decision Metric for Pulse-Jammer Scenario (Positive Received Signal Energy)

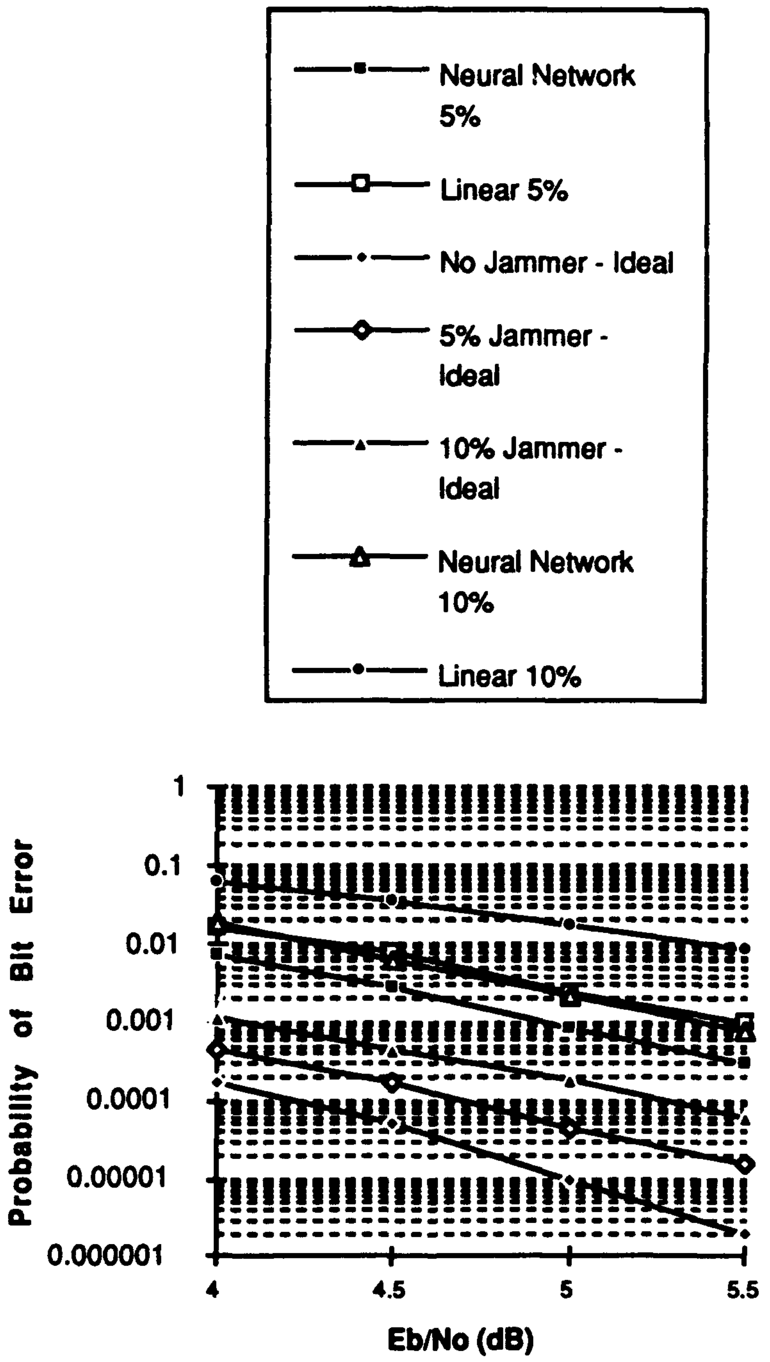


Figure 4.9-4 Bit Error Rate Performance

4.10 DEMODULATION OF QPSK WITH A RECURRENT NETWORK

Application

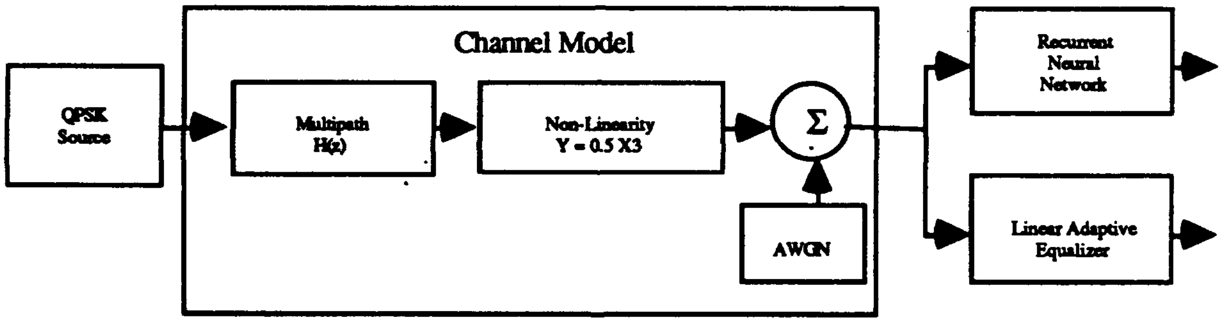
Demodulation of QPSK over a Non-Linear, Dispersive Channel Using a Fully Recurrent Network

Introduction

We examined the use of the Fully Recurrent Network in a situation where previously the Backpropagation Network was applied (in Section 4.5). In particular, the areas of channel equalization show potential as areas for Fully Recurrent application. Just as Decision Feedback techniques enhance the performance of Linear Adaptive Equalizers, we expect the Recurrent Network to be superior to Backpropagation when used for channel equalization, whether linear or non-linear. This is due to the Fully Recurrent Network architecture which contains feedback from the output layer back to the input layer.

Overview

QPSK was transmitted over a dispersive, discrete channel using the block diagram below:



The dispersive, discrete channel was modeled by the transfer function :

$$H(z) = 0.3482 + 0.8701z^{-1} + 0.3482z^{-2}$$

where z represents a delay of 1 symbol.

Furthermore, the channel imparts a non-linearity of :

$$y = 0.5x^3$$

where x is the output of the dispersive portion of the channel model, and y is the output of the non-linearity

Finally, AWGN was added.

Network Parameters

Several network configurations were examined. The first was similar to that of the Backpropagation Network in an earlier experiment. Here, the I and Q values of each of the last three received symbols constituted input layer. The second configuration attempted to make use of the Recurrent Network's inherant feedback nature by inputting on the current received symbol. It was theorized that the network would be able to form its own representation of the multipath delay.

Paradigm	Recurrent	Recurrent
Input Nodes	6	2
Hidden Nodes	8	6
Output Nodes	2	2
Learn Rate	0.01	0.01
Momentum	0.5	0.5
Update Interval	10	10
Number of Passes	100K, 100K *	100K, 100K *
Training Size	1M, 1M *	1M, 1M *
SPW Iterations per Vector	1	1

* Each of the Recurrent configurations were trained in two stages. The first stage of training was with *forced* learning, i.e., the target vector is fed back to the input layer instead of the actual output vector. The second stage of training did not involve *forced* learning. This technique is necessary since the Recurrent network uses its own output as input to the hidden layer (and also output layer) nodes. Initially the Recurrent output is very error-prone and training will not occur if the nodes are given meaningless input.

Results

A performance comparison was made between the Linear Adaptive Equalizer and the Recurrent Neural Network in terms of Bit Error Rate, showing a distinct improvement in favor of the Neural Network. The Linear Equalizer was unable to compensate for the channel distortion regardless of noise power level while the Neural Equalizer was able to significantly reduce the channel effects and result in a much lower BER.

As in the Backpropagation application to this problem, the BER curves show that several methods of applying a Neural Network to this problem:

1. Train the Neural Network to an intermediate or expected value of noise power, then fix the weights.

2. Train the Neural Network on a noiseless case, then fix the weights.

3. Continuously train the Neural Network as the noise power varies.

The BER curves corresponding the the first method described above are shown in Figure 4.10-1 for each of the Recurrent Network Configurations. A BER curve for the third method would most likely be superior to both of the other methods for all ranges of noise power. The second configuration did not perform as well as the first, but performed well considering it uses only the current received symbol as input. BER curves from the Backpropagation application to this problem are included for further comparison.

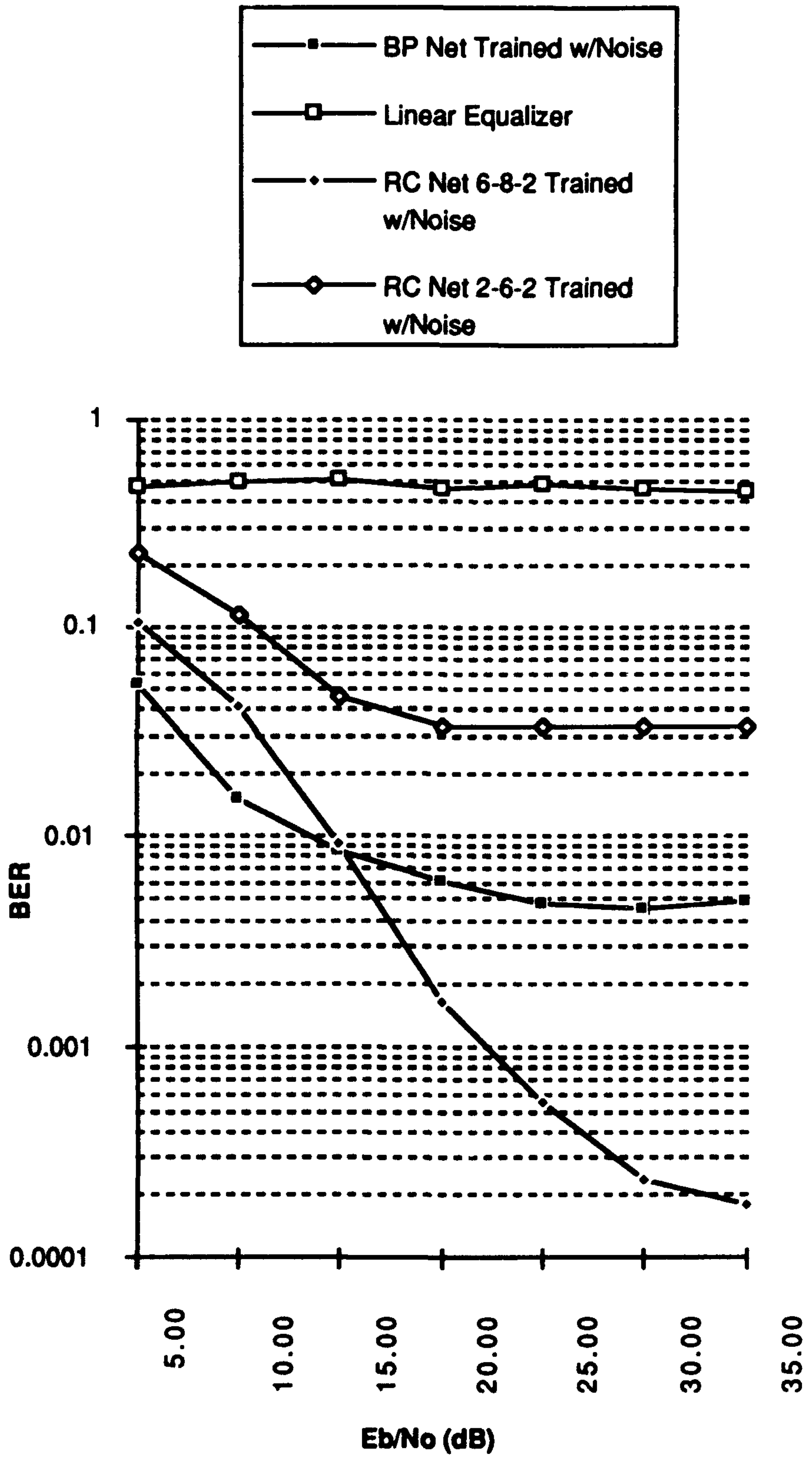


Figure 4.10 Bit Error Rate Performance

5. FUTURE NEURAL NETWORK TRANSCEIVER

A high-level block diagram of a generic multiband transceiver is shown in Figure 5-1. It is shown to illustrate where we are likely to insert neural network technology in the future. The most likely places are in the Programmable DSP Module where baseband signal processing is performed and in the Band Switch Controller where adaptive control is implemented to sense the channel conditions and switch to another band (if the frequencies are jammed or heavily used). Multiple RF modules may be employed to cover different bands. The RF modules along with the TRANSEC (if needed) and Frequency Synthesizer are implemented with conventional technology. When we develop the Phase II conceptual design, it will, of course, be much more detailed and correspond more closely to the Speak Easy radio.

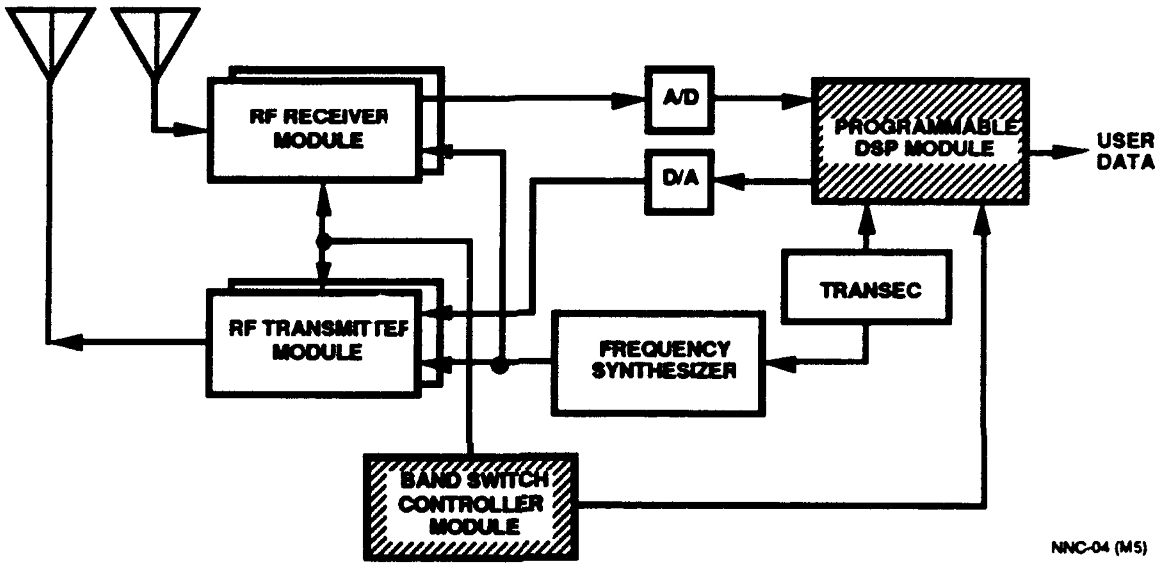


Figure 5-1 High-Level Block Diagram of Generic Multiband Transceiver (Shaded Blocks Have Potential for Utilization of Neural Network Technology).

Certain candidate problems to be addressed by neural network technology (interference cancellation, intersymbol interference elimination, multipath combining, etc.) would normally be implemented conventionally via an algorithm on the Programmable DSP Module. A neural network solution for any or all of these problems could be implemented in software on one of the DSP processors, or a neural network hardware implementation may be more desirable because of significantly increased processing power and fault tolerance. (SAIC is currently developing a neural network VLSI chip under DARPA contract which can be used for this purpose.) The tradeoffs that will be done in considering software versus hardware implementations will be the subject of one of the tasks that would make up a Phase II Neural Network Transceiver Program.

The (Phase I) Neural Network Communications Signal Processing Program (NNCSP) has addressed the question of which (or what) communications signal processing functions should be considered for implementation in a Neural Network Transceiver. Because communications signal processing is a very mature technology a greater payoff is likely if neural network implementations are considered only for those functions for which greater performance flexibility may be obtained or there is a processing speed and fault tolerance advantage provided by a highly parallel neural network implementation. Problems identified in Phase I for which this may be true include:

- Interference cancellation or mitigation
- Intersymbol interference elimination
- Multipath combining
- Joint optimization of interference cancellation, intersymbol interference elimination, and multipath combining
- Recognition of modulation type for an unknown waveform.

In addition to these problems there may be a role for neural network technology in mechanisms for adaptive data rate selection and adaptive band selection. However, the focus of the Phase I investigation was on the signal processing functions which are part of the chain of transmit and receive functions: source encoding, encryption, error control encoding, modulation, demodulation, error control decoding, decryption, and source decoding. Further, practical considerations excluded consideration in this program of source encoding and decoding and of recognition of modulation type. Within the scope of this program, neural networks were demonstrated to be effective in: eliminating intersymbol interference, multipath combining, removing nonlinear distortion, and reducing transient interference.

While conventional signal processing has been employed to accomplish all of these functions within specific environments, there is one category of comparison in which neural networks provide considerable advantage. This is the category involving flexibility, adaptivity, and robustness. In this case a neural network might not perform any better over any narrowly defined range of application, but instead might maintain the same or roughly the same performance over a significantly broader range of application than is possible with any conventional signal processing technique. Thus the metric of interest in this case measures the range of input or environmental variation over which an acceptable level of performance can be maintained. Furthermore, conventional systems may accomplish a required flexibility or robustness by employing a "man in the loop," and in that case a neural network may reduce or even eliminate the need for manual intervention in some functions.

The future Neural Network Transceiver can take advantage of improved flexibility, adaptivity and robustness by embedding neural network technology in the programmable DSP module which is highlighted in Figure 5-1. The results of Section 4 showed that existing neural network technology can provide these benefits by using neural networks for linear and nonlinear equalization, signal detection, and the generation of soft decision metrics.

A general, high-level conceptual design for implementing the functionality of the programmable DSP module is illustrated in Figure 5-2. This particular conceptual design combines and implements the neural network based adaptive signal processing functions that were simulated during this program and that were described in Section 4. Specifically, these functions are: a) equalization to correct for intersymbol interference, b) equalization to correct for nonlinear amplitude and phase distortion, c) symbol detection, and d) generation of soft bit decisions for a Viterbi error-correction decoder. In Figure 5-2, functions (a) and (b) are grouped together in one functional block and functions (c) and (d) are grouped together in another functional block. This particular grouping was chosen for purposes of conceptual description and is not meant to constrain the detailed design of these functions.

The functions are each separated into two parts: a forward processing path which does not include training and a delayed trainable path that uses the error-corrected bits from the Viterbi decoder as its target information. The adaptive weights that are trained in the delayed path are transferred to the corresponding slaved function in the forward processing path. This type of configuration was demonstrated for soft bit decisions in section 4.9.

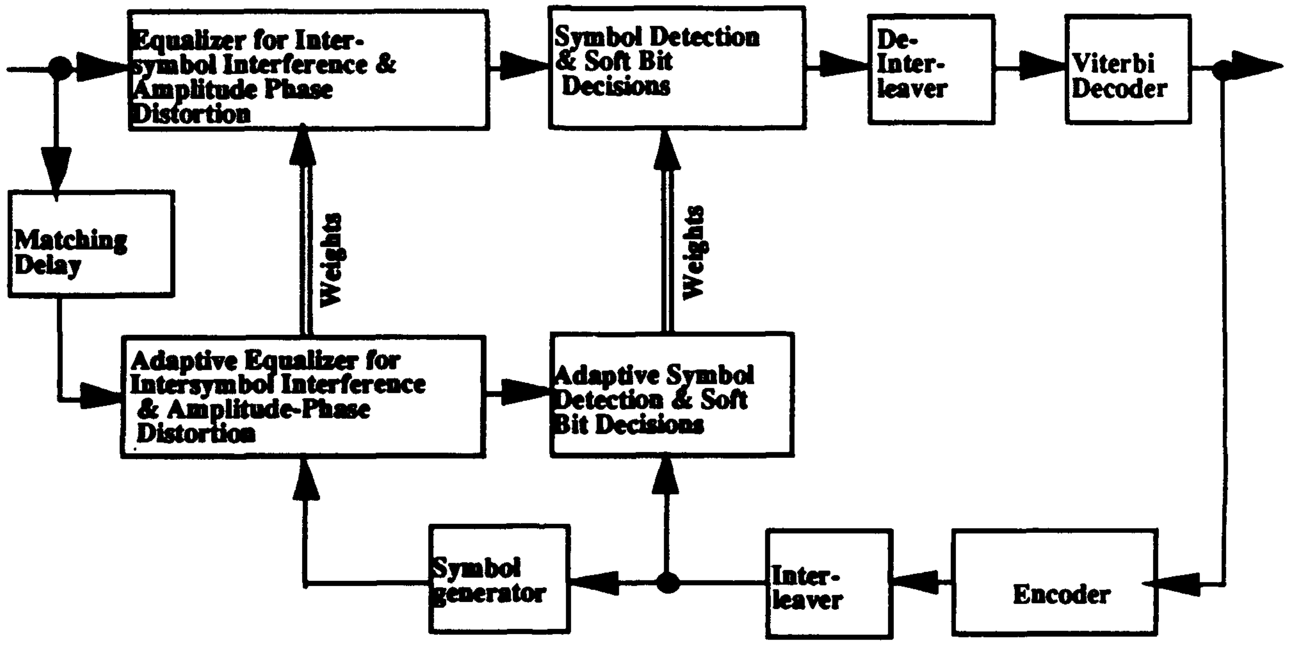

Equalizer for Inter-symbol Interference & Amplitude Phase Distortion
Symbol Detection & Soft Bit Decisions
De-Inter-leaver
Viterbi Decoder
Matching Delay
Weights
Weights
Adaptive Equalizer for Intersymbol Interference & Amplitude-Phase Distortion
Adaptive Symbol Detection & Soft Bit Decisions
Symbol generator
Inter-leaver
Encoder

Figure 5-2 Future Neural Network Receiver

6. CONCLUSIONS

The objectives of the Neural Network Communications Signal Processing (NNCSP) Program were all successfully accomplished. Specifically, the achieved objectives of the NNCSP Program are: 1) the development and implementation of a neural network and communications signal processing simulation system for the purpose of exploring the applicability of neural network technology to communications signal processing, 2) the demonstration of several configurations of the simulation to illustrate the system's ability to model many types of neural network based communication systems, and 3) the use of the simulation to identify neural network configurations to be included in the conceptual design of a neural network transceiver that could be developed in a phase II follow-on program.

The overall goal that unites the Program objectives and gives purpose to their accomplishment is to reach a new plateau in the state of the art of neural network based communications signal processing (CSP). The state of the art that existed at the start of this Program can be characterized as a collection of isolated research efforts that resulted in publications which typically described the capability and performance of one specific neural network approach to CSP. The capabilities of neural networks in a number of different CSP applications had been demonstrated, but--with few exceptions--those capabilities were not compared quantitatively with those of conventional (non-neural-network-based) state-of-the-art CSP techniques, and furthermore the publications in most cases did not give sufficient information for other researchers to reproduce the results or to use the results as a foundation upon which to build.

The NNCSP Program provides tools and techniques which can be used by future researchers to easily compare neural network and conventional techniques and to easily exchange implementation information with other researchers. The Neural Network Communications Simulation System (NNCSS) provides a block diagram approach to constructing CSP configurations for simulation, and both conventional and neural network function blocks can be used in the design and simulation of CSP products. The NNCSS provides the capability to interchange neural network modules with similar conventional signal processing modules, and makes it convenient to compare the performance of neural network based approaches with conventional approaches within the same overall system. Furthermore the NNCSS block diagram provides implementation documentation that can be archived in both paper and electronic forms. By saving the simulation block diagram file in a User Library, a researcher can automatically provide the means by which to reproduce his or her results at a later time. By sharing User Library files, researchers can easily share the implementation details that allow other researchers to reproduce their results.

As part of the NNCSP Program, ten different configurations of neural networks were simulated using the NNCSS, and the results are documented in Section 4 of this report. Nine of those simulations demonstrated neural network approaches to CSP, and seven of those demonstrated capabilities which go beyond what is currently being implemented with conventional technology. These simulations demonstrate the power and versatility of the NNCSS, and they demonstrate the adaptive nonlinear capability of neural networks in communications signal processing.

The state of the art of conventional CSP includes the capability of adaptive ***linear*** processing and some ***fixed*** nonlinear processing, but the design of conventional CSP is still typically limited by assumptions of linear channels and Gaussian interference. Neural networks go beyond conventional CSP by providing the capability of adaptive nonlinear processing which is continually self-adjusted to minimize the effects of non-Gaussian interference. In the results reported in section 4, neural networks were shown to be effective in several CSP applications: equalization to correct for intersymbol interference, equalization to correct for nonlinear amplitude and phase distortion, symbol detection for time-varying channels, and the generation of soft bit decisions for a pulse jamming environment.

Following upon the results of this Program, there are numerous opportunities for further research and development. The recommendations for further research can be grouped into three categories: further refinement and simulation of neural network based CSP applications, further improvements to the NNCSS, and the development of a prototype neural network based transceiver.

Additional opportunities exist for refining and extending the applications summarized in Section 4, and there are many more applications that have been described in the literature which can be further refined and compared to conventional approaches using the NNCSS. While it is not practical to name here all of the potentially useful techniques that could be investigated, one particular area of development is worth mentioning here because it is a continuation of the application described in 4.9. In particular, the neural network based generation of soft decisions can be extended to additional modulation formats and, in addition to pulsed jamming, the technique is applicable to atmospheric noise, and to the near-far interference problem of code-division multiple-access. The recent article by Asato and Grover [Asato,1993] presents performance bounds which suggest that very significant improvements in performance can be obtained.

The NNCSS in its initial deliverable version is a remarkably versatile and capable design and simulation tool. Nevertheless, several incremental improvements can be recommended. The first set of recommendations would be to add as options several learning acceleration techniques for backpropagation such as delta-bar-delta, the Levenberg-Marguardt algorithm, and the entropic error function. The second recommendation would be to add the Radial-Basis Function Neural Network as an additional function block.

This Program was strategically positioned and specifically aimed at the prerequisites needed prior to a follow-on program to develop a neural network transceiver. The necessary design and simulation tools have been incorporated in the NNCSS, and the selection and simulation of potential neural network based CSP functions are documented in Section 4. Section 5 presents a high-level, conceptual design of a future neural network transceiver which is recommended for a follow-on program.

7. BIBLIOGRAPHY

Aazhang, Behnaam, Paris, Bernd-Peter, Orsak, Geoffrey C., "Neural Networks for Multiuser Detection in Code-Division Multiple-Access Communications," *IEEE Transaction on Communications,* Vol. 40, No. 7, July 1992 pp. 1212-22.

Almeida, L.B., "A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment," *Proceedings of the 1987 IEEE Conference on Neural Networks,* Vol. 2, 1987.

Anderson, J.A. and Mozer, M.C., "Categorization and Selective Neurons," In G.E. Hinton and J.A. Anderson, Eds. *Parallel Models of Associative Memory* (Rev. Ed.), Erlbaumm Hillsdale, NJ, 1989, pp. 251-276.

Anderson, J.A., Silverstein, J.W., Ritz, S.A. and Jones, R.S., "Distinctive Features, Categorical Perception, and Probability Learning: Some Applications of a Neural Model," *Psychological Review,* Vol. 84, 1977, pp. 413-451.

Anderson, James A., Gately, Michael T., Penz, P.A., and Collins, Dean R., "Radar Signal Categorization Using a Neural Network," *Proc. of the IEEE,* Vol. 78, No. 10, October 1990, pp. 1646-1657.

Andersson, Gunnar, Andersson, Hakan, "Generation of Soft Information in a Frequency-Hopping HF Radio System Using Neural Networks," *Milcom '92, Communications - Fusing Command, Control and Intelligence,* Volume 2, Session 33A, pp. 779-783.

Asato, S. , Grover, M.K., and Cahn, Charles R., "Artificial Neural Network Adaptive Nonlinear Digital Receivers," *Government Microcircuit Applications Conference Digest of Papers,* 1993, pp. 257-260.

Baehre, Mark D., "Neural Networks Applied to Signal Processing," *DTIC,* September 1989.

Barnhart, Craig M., Wieselthier, Jeffrey E., Ephremides, Anthony, "Scheduling Link Activation in Multihop Radio Networks By Means of Hopfield Neural Network Techniques," *DTIC,* Sep. 03, 1991.

Berman, Piotr, Schnitger, George, Parberry, Ian, "A Complexity Theory of Neural Networks," *DTIC,* Aug. 09, 1991.

Bijjani, R., and Das, P.K., Rejection of Narrowband Interference in PN Spread-Spectrum Systems Using Neural Networks", *IEEE Global Telecommunications Conference & Exhibition Part 2 of 3,* Dec. 2-5, 1990, pp. 1037-1041

Cain, G.D., Yardim, A., and Taori, R., "Error Measurement Issues in Darwinian Adaptive Notch Filtering," *IEEE International Workshop on Intelligent Signal Processing and Communication Systems,* Taipei, Taiwan, R.O.C., March 1992.

Carpenter, G.A. and Grossberg, S., "Absolutely Stable Learning of Recognition Codes by a Self-Organizing Neural Pattern Machine," *Computer Vision, Graphics, and Image Processing,* 1986.

Carpenter, G.A. and Grossberg, S., "ART 2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns," *Applied Optics*, Special Issue on Neural Networks, Vol. 26, 1987, pp. 4919-4930.

Carpenter, Gail A. and Grossberg, Stephen, "ART 3: Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures," *Neural Networks,* Vol. 3, 1990, pp. 129-152.

Carpenter, Gail A., Grossberg, Stephen, Rosen, David B., "ART 2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition," *Neural Networks,* Vol. 4, 1991, pp. 493-504.

Carpenter, Gail A., "Neural Network Models for Pattern Recognition and Associative Memory," *Neural Networks,* Vol. 2, 1989, pp. 243-257.

Chang, Po-Rong, Yang, Wen-Hao, and Chan, Kuan-Kin, "A Neural Network Approach to MVDR Beamforming Problem," *IEEE Transactions on Antennas and Propagation*, Vol. 40, No. 3, March, 1992, pp. 313-322.

Chapman, Richard A., "Classification of Correlation Signatures of Spread Spectrum Signals Using Neural Networks," *DTIC*, January, 1991.

Chen, Sheng, Mulgrew, Bernard, and Grant, Peter M., "A Clustering Technique for Digital Communications Channel Equalization Using Radial Basis Function Networks," *IEEE Transactions on Neural Networks,* Vol. 4, No. 4, July 1993, pp. 570-579.

Chesmore E.D., "Application of pulse processing neural networks in communications and signal demodulation," *First IEEE International Conference on Artificial Neural Networks* (Conf. Publ. No. 313) pp. 337-341.

Choi, Joongho, Bang, Sa Hyun, and Sheu, Bing J., "A Programmable Analog VLSI Neural Network Processor for Communication Receivers," *IEEE Transactions on Neural Networks,* Vol. 4, No. 3, May 1993, pp. 484-495.

Crooks, Ted, "Care and Feeding of Neural Networks," *AI Expert,* July 1992, pp. 37-41.

de Veciana, Gustavo, Zakhor, Avideh, "Neural Net-Based Continuous Phase Modulation Receivers," *IEEE Transaction on Communications,* Vol. 40, No. 8, August 1992, pp. 1396-1408.

Doner, John R., "Improving Unsupervised Learning in the Neocognitron", *SIMTECH '91 Proceedings, Society for Computer Simulation*, Orlando, FL, October, 1991, pp. 589-594.

Dubosq, Philippe, "An Overview of Artificial Intelligence," *Brevard Technical Journal,* August 1993, pp. 47-49.

Dubosq, Philippe, "An Overview of Artificial Intelligence Part II: Neuron Activation versus Symbol Manipulation," *Brevard Technical Journal,* Sept. 93, pp. 35-39.

Feiz, S., Soliman, A. S., "Adaptive ML neural network based receiver for Q^2PSK modulated data-transmission systems," *39th IEEE Vehicular Technology Conference* (IEEE Cat. No. 89CH2739-1) pp. 263-9 Vol. 1.

Field, R. L., and Yoerger, E. J., "Performance of Neural Networks in Classifying Environmentally Distorted Transient Signals," *IEEE Conference Proceedings - OCEANS '90*, Sept. 24-26, 1990, pp. 144-148.

Fontana, Robert J., Mort, Michael S., "Communications Signal Recognition and Demodulation via Neural Networks," *DTIC*, May, 1991.

Freeman, James A. and Skapura, David M., *Neural Networks*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1991.

Garcia-Gomez, R., Gomez-Mena, J., and Diez del Rio, L., "Adaptive Receivers for Removing Linear and Non-linear Intersymbol Interference by Mean of Time Delay Neural Nets (AR-TDNN)," *1989 International Conference on Acoustics, Speech and Signal Processing*, May 23-26, 1989, pp. 2368-2371.

Hancock, Monte F., Jr., MS., "Solving Hard Problems Using Machines That Learn," *Brevard Technical Journal*, August, 1992, pp. 23-24, 41-45.

Haykin, Simon, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Company, New York, 1994.

He, Zhen-Ya, and Li, Li-Hua, "High-Resolution Multipath Time Delay Estimation Using a Neural Network," Proceedings of the 1991 International Conference on Acoustics, Speech, and Signal Processing - ICASSP 91, May 14-17, 1991, pp. 1469-1472.

Hecht-Nielsen, Robert, "Applications of Counterpropagation Networks," *Neural Networks*, Vol. 1, pp. 131-139, 1988

Hecht-Nielsen, Robert, *Neurocomputing*, Addison-Wesley, Reading, MA, 1990.

Hiramatsu, Atsushi "Integration of ATM Call Admission Control and Link Capacity Control by Distributed Neural Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 7, September 1991 pp. 1131-38.

Horner, Larry, Madey, Greg, Sandridge, Brian, "Application of Artificial Neural Nets for Recognition of Forward Error Coding," *DTIC*, Aug 31, 1991.

Hussain, M., Jing Song, Bedi, J.S., "Neural network application to error control coding," *Proc of the SPIE - The International Society for Optical Engineering* vol. 1294 pp. 502-9.

Hyland, David C., King, James A., "Neural Architectures for Stable Adaptive Control, Rapid Fault Detection and Control System Recovery," *Harris White Paper*, 1992

Jacobs, R.A., "Increased rates of convergence through learning rate adaptation," *Neural Networks*, Vol 1, 1988, pp. 295-307.

Jake, W.C., *Microwave Mobile Communication*, John Wiley & Sons, New York, 1974.

Jeffries, C., Protzel, P., "High-order neural models for error correcting code," *Proc of the SPIE - The International Society for Optical Engineering* Vol. 1294 pp. 510-17.

Johnson, J.R., "Neural network algorithm decoding and sequence predictor," *INNC 90 Paris. International Neural Network Conference* pp. 153-6 Vol. 1.

Kechriotis, G., Zervas, E., and Manolakos, E.S., "Using Recurrent Neural Networks for Adaptive Communication Channel Equalization," *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, March 1994, pp. 267-278.

Kechriotis, G., Zervas, E., Manolakos, E. S., "Using Recurrent Neural Networks For Blind Equalization Of Linear and Nonlinear Communication Channels," *Milcom '92, Communications - Fusing Command, Control and Intelligence,* Volume 2, 1992, pp. 784-788.

Kim, Myung Soo and Guest, Clark C., "Modification of Backpropagation Networks for Complex-Valued Signal Processing in Frequency Domain," *International Joint Conference on Neural Networks*, 1990, pp. III-27–III-31.

King, Todd, "Using Neural Networks for Pattern Recognition: Recognizing and Learning patterns is one thing neural nets do best," *Dr. Dobb's Journal,* January, 1989, pp. 14-28.

Klimasauskas, Casey, "Neural Nets and Noise Filtering: Back-propagation is a powerful adaptive method for filtering out noise or identifying underlying signals," *Dr. Dobb's Journal,* January, 1989, pp. 32-48.

Kohonen, Teuvo, Raivio, Kimmo, Simula, Olli, and Henriksson, Jukka, "Performance Evaluation of Self-organizing Map Based Neural Equalizers in Dynamic Discrete-Signal Detection," *Proceedings of the 1991 International Conference on Artificial Neural Networks (ICANN-91) Volume 2*, June 24-28, 1991, pp. 1677-1680.

Kohonen, Teuvo, Raivio, Kimmo, Simula, Olli, Venta, Olli, Henriksson, Jukka, "Combining Linear Equalization and Self-Organizing Adaptation in Dynamic Discrete Signal Detection", *International Joint Conference on Neural Networks* , 1990, vol. 1, pp. I-223 - I-228.

Kohonen, Teuvo, *Self-Organizing and Associative Memory,* Springer-Verlag, Series in Information Sciences, Vol. 8, Berlin-Heidelberg-New York-Tokyo, 1984.

Kovalick, Al and Titchener, Paul, "Bridge the Gap Between Simulation and Hardware Prototyping," *Electronic Design*, June 13, 1991, pp. 77-88.

Kunz, Dietmar, "Channel Assignment for Cellular Radio Using Neural Networks," *IEEE Transactions on Vehicular Technology,* Vol. 40, No. 1, February, 1991. pp. 188-193.

Lawrence Jeannette and Andriola, Peter, "Three-step method evaluates neural networks for your application," *EDN* August 6, 1992, pp. 93-100.

Lee, Tsu-Chang, Peterson, Allen M., "Adaptive Vector Quantization Using a Self-Development Neural Network," *IEEE Journal on Selected Areas in Communications,* Vol. 8, No. 8, October 1990, pp. 1458-71.

Lippmann, Richard P., "An Introduction to Computing with Neural Nets, " *IEEE ASSP Magazine,* April 1987, pp. 4-22.

Lirov, Yuval, "Computer Aided Neural Network Engineering," *Neural Networks,* Vol. 5, pp. 711-719, 1992.

Naylor, J.A., "A neural network algorithm for enhancing delta modulation/LPC tandem connections," *ICASSP 90. 1990 International Conference on Acoustics, Speech and Signal Processing* (Cat. No. 90CH2847-2) pp. 221-4 Vol. 1.

Nguyen, Derrick and Widrow, Bernard, "Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights," *International Joint Conference on Neural Networks,* 1990, pp. III-21–III-26.

Nobakht, Ramin A., Van Den Bout, David E., Townsend, J. Keith, Ardalan, Sasan H., "Optimization of Transmitter and Receiver Filters for Digital Communications Systems Using Mean Field Annealing," *IEEE Journal on Selected Areas in Communications,* Vol. 8, No. 8, October, 1990, pp. 1472-80.

Peng, Marcia, Nilias, Chrysostomos L., and Proakis, John G., "Adaptive Equalization for PAM and QAM Signals," *Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems, & Computers,* Volume 1 of 2, November 4-6, 1991, pp. 496-500.

Raeth, Peter G., "Event-Train Restoration Via Back propagation Neural Networks," *DTIC,* Dec., 1989.

Rafie, Manouchehr S. and Shanmugan, K. Sam, "Simulation of an End-to-End 9600 bps V.32 Modem Using SPW," *Proceedings of the 1991 International Simulation Technology Conference (SIMTEC '91),* Orlando, Florida, October 1991.

Rao, Suthyanarayan S., Sethuraman, Sriram, "A Neural Network Tunable Filter For Multi-Tone Detection," *Milcom '92, Communications - Fusing Command, Control and Intelligence,* Volume 2, Session 33A, pp. 789-793.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J., "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing,* Vol. 1, Bradford Books - The MIT Press, 1986.

Rummler, William D., "More on the Multipath Fading Channel Model," *IEEE Transactions on Communications,* Vol. COM-29, No. 3, March 1981, pp. 346-352.

Santamaria, M.E., Lagunas, M.A., Cabrera, M., "Neural nets filters: integrated coding and signaling in communication systems," *MELECON '89: Mediterranean Electrotechnical Conference Proc. Integrating Research, Industry and Education in Energy and Communication Engineering* (Cat. No. 89CH2679-9) pp. 532-5.

Sejnowski, Terrence, "IEEE Conference on Neural Information Processing Systems - Natural and Synthetic Held In Denver, Colorado on 28 November - 1 December 1988," *DTIC,* Aug., 14, 1989.

Sengoku, M., Nakano, K., Shinoda, S. Yamaguchi, Y. Abe, T. "Cellular mobile communication systems and a channel assignment using neural networks, "*Proc of the 33rd Midwest Symposium on Circuits and Systems* (Cat. No. 90CH2819-1) pp. 411-14 Vol. 1.

Sietsma, Jocelyn and Dow, Robert J.F., "Creating Artificial Neural Networks That Generalize," *Neural Networks,* Vol. 4.,1991, pp. 67-79.

Siu, S., Gibson, G.J., Cowan, C.F.N., "Decision feedback equalization using neural network structures," *First IEE International Conference on Artificial Neural Networks* (Conf. Publ. No. 313) pp. 125-8.

Sorensen, Helge B. D., "A Cepstral Noise Reduction Multi-Layer Neural Network," *ICASSP 91, 1991 International Conference on Acoustics, Speech and Signal Processing, Vol. 2,* pp. 933-6.

Specht, D.F. and Shapiro, P.D., "Generalization Accuracy of Probabilistic Neural Networks Compared with Backpropagation Networks," *Proceedings of the International Joint Conference on Neural Networks*, IEEE Press, New York, 1991.

Specht, D.F., "Probability Neural Networks and the Polynomial Adaline as Complementary Techniques for Classification," *IEEE Transactions on Neural Networks*, Vol. 1, 1990.

Specht, Donald F., "Probabilistic Neural Networks," *Neural Networks,* Vol. 3, 1990, pp. 109-118.

Speidel, S. L., "Neurobeamformer II: Further Exploration of Adaptive Beamforming via Neural Networks," *DTIC*, Nov. 22, 1989.

Stubbendieck, G.T., Oldham, W.J.B. "Recognition of patterns in electronic communication signals using neural networks," *Knowledge-Based Systems and Neural Networks: Techniques and Applications* pp. 237-51.

Tasic, J. Grlj, M. "Theory and application of the neural net based adaptive filter in communication systems," *Communication, Control and Signal Processing. Proc of the 1990 Bilkent International Conference on New Trends in Communication, Control and Signal Processing* pp. 1788-95 Vol. 2.

Thacker, Neil, A., Mayhew, John E.W., "Designing a Layered Network for Context Sensitive Pattern Classification," *Neural Networks,* Vol. 3, 1990, pp. 291-299.

Van Ooyen., A. and Nienhuis, B. "Improving the Convergence of the Back-propagation Algorithm, "*Neural Networks*, Vol. 5, 1992, pp. 465-471.

Welstead, Stephen T., Ward, Michael J., and Keefer, Christopher W., "Neural Network Approach to Multipath Delay Estimation," *SPIE Vol. 1565 Adaptive Signal Processing*, 1991, pp. 482-491.

Wieselthier, J. E., Barnhart, C. M., and Ephrimedes, A., "Sequential Link Activation in Multihop Radio Networks by Means of Hopfield Neural Network Techniques," *Proceedings of the 1991 International Symposium on Information Theory,* p. 155, June 1991.

Williams, R.J. and Zipser, D., "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation,* 1, 1989, pp. 270-280.

Xiang, Zengjun, and Bi, Guangguo, "New Fractionally Spaced Recursive Polynomial Perceptron Model for Adaptive M-QAM Digital Mobile Radio Reception," *Electronic Letters*, Vol. 28, No. 22, Oct. 22, 1992, pp. 2049-2051.

Ziemer, Rodger E. and Peterson, Roger L., *Digital Communications and Spread Spectrum Systems*, Macmillan Publishing Company, New York, 1985.

8.0 GLOSSARY

ABAM	Adaptive Bi-directional Associative Memory
ANN	Artificial Neural Network
ANS	Artificial Neural System
ART	Adaptive Resonance Theory
AWGN	Additive White Gaussian Noise
BAF	Block Attributes File
BAM	Bi-directional Associative Memory
BER	Bit Error Rate
BEX	Block EXpression file (binary)
BDE	Block Diagram Editor
BP	Backpropagation
BPSK	Binary Phase Shift Keying
BSB	Brain State in a Box
CCFB	Custom Coded Function Block
CDRL	Contract Data Requirements List
CGS	Code Generation System
COTS	Commercial Off The Shelf
CPU	Computer Processing Unit
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSCI08	Computer Software Configuration Item No. 8 (NNCL)
CSP	Communications Signal Processing
CSU	Computer Software Unit
CUFB	Custom User Function Block
CUPS	Connection Updates Per Second
DI	Developmental Item
DID	Data Item Description
DMA	Direct Memory Access
DOD	Department of Defense
DSP	Digital Signal Processing
EXPR	EXPRession file (text).
FIR	Finite Impuse Response
FFT	Fast Fourier Transform
FMS	File Management System
GFI	Government Furnished Item
GUI	Graphical User Interface
HF	High Frequency
HWCI	Hardware Configuration Item
ISI	Inter Symbol Interference
ISL	Interactive Simulation Library™

ISNN	Industrial Strength Neural Network
KTM	Kohonen Topological Map
LMS	Least Mean Squared
LTM	Long Term Memory
MCL	Macro Command Language
MHz	Mega (Million) Hertz (cycles per second)
MIPS	Million Instructions Per Second
ML	Maximum Likelihood
MMI	Man/Machine Interface
NDI	Non-Developmental Item
NNCL	Neural Network Communications Library
NNCSP	Neural Network Communications Signal Processing
NNCSS	Neural Network Communications Simulation System
NNO	Neural Network Object
NNOC	Neural Network Object Control
NNOM	Neural Network Object Manager
PN	Probabilistic Network
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
RC	Recurrent
RF	Radio Frequency
RFP	Request for Proposal
RISC	Reduced Instruction Set Computer
RLS	Recursive Least Squares
RMS	Root Mean Square
SFS	Signal Flow Simulation
SigCalc	Signal Calculator™
SOFM	Self Organizing Feature Map
SPB	Simulation Program Builder
SPW	Signal Processing WorkSystem™
SSDD	System/Segment Design Document
SDD	Software Design Document
STM	Short Term Memory
TIL	Tool Interface Language™
TWT	Traveling Wave Tube
VLSI	Very Large Scale Integration

APPENDIX

PROCEDURES FOR RUNNING THE SIMULATIONS

These instructions give steps to execute the simulations described in Section 4. It is assumed that SPW and the associated NNCSS tapes have been installed and a simulation kernel in nncss_all has been created. In each of the following ten simulation procedures "Run the simulation for XXX iterations" will mean to perform the following operations:

Select Tools-Simulator-Run.

Select More Options

Enter (or select from the extended dialog button) nncss_all as the Simulation Kernel.

Press OK

Enter XXX in the No. of Iterations field

Press Start

1. Non-Linear Mapping by a Backpropagation Neural Network (refer to 4.1)

Step 1 Make sure that weights are initialized from random values instead of from the stopping point of a previous execution of this system: rm /spwsys/pool/nncss_all/sqr.net

Step 2 Open the SPW simulation model entitled square(10).system.

Step 3 Run the simulation for 10K iterations. This will display the network learning the function $0.5x^3$.

Step 4 Change the value of the constant feeding into the y-input of the x^y block to 2.0. This will display the network learning the function $0.5x^2$, starting with the weights which resulted from training the network to learn the $0.5x^3$ function.

Step 5 Select File-Close on the Simulation Run window.

2. Equalization of Multipath Distorted 64-QAM using a Backpropagation Neural Network (refer to 4.2)

This demonstration displays the results of a trained Backpropagation Network to equalize a Multipath-Distorted 64-QAM signal. The actual training of the network is time-consuming and is not appropriate for an interactive demonstration using the Interactive Simulation Library (ISL). To execute the ISL demonstration:

Step 1 Since this demonstration displays results for a previously trained network, we must begin with the weights which resulted from this earlier training: cp /spwsys/pool/tbill_3_all/64qam_3.net /spwsys/pool/nncss_all/64qam_3.net

Step 2 Open the SPW simulation model entitled bp_eval(27).system.

Step 3 Turn learning off by changing the value of the constant feeding into the *train* input to the bpnet block to 0.0.

Step 4 Run the simulation for 30K iterations.

Step 5 Select File-Close on the Simulation Run window.

To see that the neural network can indeed be trained to equalize a Multipath-Distorted 64-QAM signal, this simulation can be executed in a non-ISL mode with the following modifications:

Step 1 Make sure that weights are initialized from random values instead of from the stopping point of a previous execution of this system: rm /spwsys/pool/nncss_all/64qam_3.net

Step 2 If it is not already on, turn learning on by changing the value of the constant feeding into the *train* input to the bpnet block to 1.0.

Step 3 To expedite the simulation, select and cut all of the ISL blocks in the ISL Output portion of the system diagram.

Step 4 In the Network and Output Control portion of the system diagram, edit the parameter on the Unit Step block which feeds into the Inverter and *wait* connector. Change its value to 950000. This will cause signal files to be written to disk only after simulation iteration 950000 has been reached, thus writing less to disk.

Step 5 Run the simulation for 1000000 iterations. This may take about 90 minutes to complete.

Step 6 After completion, press SigCalc on the Simulation window to view resulting signals.

Step 7 Select File-Close on the Simulation Run window.

3. Use of a Backpropagation Neural Network to Control a Bank of Equalizers in a Dynamic Multipath Environment (refer to 4.3)

This demonstration displays the results of a trained Backpropagation Network to equalize a Dynamic Multipath-Distorted 64-QAM signal using the outputs of a bank of equalizers. The actual training of the network is time-consuming and is not appropriate for an ISL demonstration. To execute the ISL demonstration:

Step 1 Since this demonstration displays results for a previously trained network, we must begin with the weights which resulted from this earlier training: cp /spwsys/pool/tbill_3_all/mmpath_1.net /spwsys/pool/nncss_all/mmpath_1.net

Step 2 Open the SPW simulation model entitled bp_mmtest(8).system.

Step 3 Run the simulation for EOF iterations. The simulation will read test data from a file, re-starting from the beginning when the end is reached. When satisfied with the ISL display, press Abort in the Simulation Run window.

Step 4 Select File-Close on the Simulation Run window.

4. Demodulation of Non-Linearly Distorted 16 QAM Using Backpropagation (refer to 4.4)

Step 1 Make sure that weights are initialized from random values instead of from the stopping point of a previous execution of this system: rm /spwsys/pool/nncss_all/classify6.net

Step 2 Open the SPW simulation model entitled classify(16).system.

Step 3 Run the simulation for 25K iterations. You will observe the error rate of the neural network decrease drastically over the course of the simulation as the network learns better decision regions, and eventually performing better than the Linear Decision Regions for Ideal 16 QAM.

Step 5 Select File-Close on the Simulation Run window.

5. Demodulation of QPSK over a Non-Linear, Dispersive Channel using Backpropagation Neural Network (refer to 4.5)

This demonstration displays the results of a trained Backpropagation Network to demodulate a QPSK signal which has been transmitted over a non-linear, dispersive, AWGN channel . The actual training of the network is time-consuming and is not appropriate for an ISL demonstration. To execute the ISL demonstration:

Step 1 Since this demonstration displays results for a previously trained network, we must begin with the weights which resulted from this earlier training: cp /spwsys/pool/tbill_3_all/qpsk6.net /spwsys/pool/nncss_all/qpsk6.net

Step 2 Open the SPW simulation model entitled bp_qpsk(13).system.

Step 3 Make sure that learning is off by changing the value of the constant feeding into the *train* input to the bpnet block to 0.0.

Step 4 Run the simulation for 10K iterations. The in-phase and quadrature components produced by the neural network (shown in a constellation diagram) are thresholded to produce the demodulated QPSK symbol.

Step 5 Select File-Close on the Simulation Run window.

To see that the neural network can indeed be trained to demodulate a QPSK signal which has been transmitted over a non-linear, dispersive, AWGN channel , this simulation can be executed in a non-ISL mode with the following modifications:

Step 1 Make sure that weights are initialized from random values instead of from the stopping point of a previous execution of this system: rm /spwsys/pool/nncss_all/qpsk6.net

Step 2 If it is not already on, turn learning on by changing the value of the constant feeding into the *train* input to the bpnet block to 1.0.

Step 3 To expedite the simulation, select and cut all of the ISL blocks in the ISL Output portion of the system diagram.

Step 4 In the Network and Output Control portion of the system diagram, edit the parameter on the Unit Step block which feeds into the Inverter and *wait* connector. Change its value to 950000. This will cause signal files to be written to disk only after simulation iteration 950000 has been reached, thus writing less to disk.

Step 5 Run the simulation for 1000000 iterations. This may take about several hours to complete.

Step 6 After completion, press SigCalc on the Simulation window to view resulting signals.

Step 7 Select File-Close on the Simulation Run window.

6. Demodulation of QPSK using a Configuration of Kohonen and Outstar Neural Networks (refer to 4.6)

Step 1 Make sure that weights are initialized from random values instead of from the stopping point of a previous execution of this system: rm /spwsys/pool/nncss_all/koh3.net

Step 2 Open the SPW simulation model entitled koh_qpsk(8).system.

Step 3 The Kohonen and Outstar networks each are controlled by a Neural Network Object Controller (NNOC) block. The NNOC for the Outstar is on the top system level, while the NNOC for the Kohonen Network is inside the KTMNET block. The *Delta Learning Threshold* parameter inside each of these NNOC blocks should be changed to -1.0. This is necessary since there is no delta input to the NNOCs. By definition, if the *delta* signal is less than or equal to the value of the *Delta Learning Threshold* parameter, then weights will no longer be updated.

Step 4 Run the simulation for 20K iterations.

Step 5 Select File-Close on the Simulation Run window.

7. Demodulation of Non-Linearly Distorted 16 QAM using a Configuration of Kohonen and Outstar Neural Networks (refer to 4.7)

Step 1 Make sure that weights are initialized from random values instead of from the stopping point of a previous execution of this system: rm /spwsys/pool/nncss_all/koh16qam13.net

Step 2 Open the SPW simulation model entitled 16qam_twt(18).

Step 3 The Kohonen and Outstar networks each are controlled by a Neural Network Object Controller (NNOC) block. The NNOC for the Outstar is on the top system level, while the NNOC for the Kohonen Network is inside the KTMNET block. The *Delta Learning Threshold* parameter inside each of these NNOC blocks should be changed to -1.0. This is necessary since there is no delta input to the NNOCs. By definition, if the *delta* signal is less than or equal to the value of the *Delta Learning Threshold* parameter, then weights will no longer be updated.

Step 4 Run the simulation for 35K iterations.

Step 5 Select File-Close on the Simulation Run window.

8. Demodulation of 16 QAM over a Rayleigh Channel using a Configuration of Kohonen and Outstar Neural Networks (refer to 4.8)

Step 1 Make sure that weights are initialized from random values instead of from the stopping point of a previous execution of this system: rm /spwsys/pool/nncss_all/mobile.net

Step 2 Open the SPW simulation model entitled mobile(15).

Step 3 The Kohonen and Outstar networks each are controlled by a Neural Network Object Controller (NNOC) block. The NNOC for the Outstar is on the top level inside of the Kohonen Equalizer block, while the NNOC for the Kohonen Network is inside the KTMNET block, which is also inside of the Kohonen Equalizer block. The *Delta Learning Threshold* parameter inside each of these NNOC blocks should be changed to -1.0. This is necessary since there is no delta input to the NNOCs. By definition, if the *delta* signal is less than or equal to the value of the *Delta Learning Threshold* parameter, then weights will no longer be updated.

Step 4 Run the simulation for 35K iterations.

Step 5 This demonstration contains many controls which make it rather complicated. There are 3 Eye and Scatter diagrams which appear. In each case, the Eye pattern produced is to be ignored. The Eye and Scatter blocks were used simply for their built-in controls. The leftmost scatter plot is the received constellation from the Rayleigh channel. Over the course of the demonstration, it will contract, expand, and rotate. As it contracts and expands, it will become necessary to adjust the Gain via the scroll bar to keep its display within the bounds of the scatter window. The center scatter plot is the resulting constellation produced by a stand-alone Linear Adaptive Equalizer. Given a training signal, the equalizer can adjust to the Rayleigh effects. As the Rayleigh fades occur, the equalizer will lose track of the rotated/compressed/expanded constellation and again will require a training signal to re-adjust.

The rightmost scatter plot is a plot of the Kohonen weights. These weights will converge to the centers of the clusters formed by the equalizer portion of the Kohonen Equalizer. They will individually move in order to quickly track the rotating/compressing/expanding constellation.

Below the Eye and Scatter controls are several pushbuttons which control training and learning for the stand-alone Linear Adaptive Equalizer and the Kohonen Equalizer.

Train KTM - Toggles weight adjustment for Kohonen portion of the Kohonen Equalizer, whether in normal or adaptive mode.

Train Equalizer - Toggles the presentation of the actual transmitted 16 QAM signal as a training signal to the stand-alone Linear Adaptive Equalizer.

Train Outstar - Toggles the presentation of the actual transmitted 16 QAM signal as a training signal to the Outstar Network.

Use Training Sig - Toggles the presentation of the actual transmitted 16 QAM signal as a training signal to the equalizer portion of the Kohonen Equalizer.

Use Conventional Decision Regions - In the Kohonen Equalizer, an error signal is fed back to update the weights of the equalizer portion. This pushbutton controls the method of creating this error signal. If the button is pressed, the error signal is the difference between the closest Ideal QAM symbol and the equalizer output. If the button is not pressed, the error signal is the difference between the weights corresponding to the winning Kohonen node and the equalizer output.

Re-Train KTM - If a severe null occurs, the Kohonen Equalizer will be unable to track the moving symbol constellation, and will require re-training. When this button is pressed, retraining will occur. Note that re-training the Kohonen network will require a re-mapping (and hence re-training) of the Outstar Network.

At the bottom of the display are two bar graphs which display symbol errors. The left graph corresponds to symbol errors made by the Linear Adaptive Equalizer, and the right for the Kohonen Equalizer. Note that when a training signal is given to either equalizer, the corresponding symbol error graph is disabled and will show a zero value.

Once the ISL Window appears on the screen, first set the Scatter Persistence scroll bar on the rightmost scatter plot to 16, since we are interested in the weights corresponding to the 16 Kohonen nodes. Initially, all buttons except for the *Re-Train KTM* button should be depressed. Around simulation iteration 5000, the Kohonen network enters adaptive mode. At this time, the Kohonen weights have found the symbol centers of the equalized constellation and are tracking movement. The stand-alone equalizer too has been trained to equalize the received signal. *Train Equalizer* and *Use Training Signal* may be turned-off. Over the next 6000 simulation iterations, the Outstar Network will map each Kohonen node to an explicit 16 QAM symbol. The Outstar training may be accelerated by increasing the learn and decay rates. So at around iteration 11K, *Train Outstar* may be turned off. Depending upon the noise seed and the Doppler of the Rayleigh channel (the scrollbar entitled *Amplitude* on the display), the Kohonen Equalizer and the stand-alone equalizer will correctly demodulate the received signal, as evidenced by the two bar graphs at the bottom of the display. Usually, the stand-alone equalizer will fail first during appearance of nulls. Failing is indicated by a large number of spikes (corresponding to symbol errors) in the bar graphs.

Once either equalizer has failed, a training sequence is required for proper demodulation to resume. If the stand-alone equalizer fails, press the *Train Equalizer* button. The equalizer will require some time to re-adjust its weights. Once the center constellation appears fairly organized, the *Train Equalizer* button may be turned off. Note that when a training signal is given to either equalizer, the corresponding symbol error graph is disabled and will show a zero value. If the Kohonen Equalizer fails, the KTM must be redone, and hence the Outstar mapping. This is accomplished by depressing *Train KTM*, *Train Outstar, Use Training Sig*, *Use Conventional Decision Regions*, and *Re-Train KTM*. After several thousand iterations, the KTM is ready to resume adaptive mode. Toggle *Use Training Sig*, *Use Conventional Decision Regions*, and *Re-Train KTM*. The Kohonen weight should now appear at the 16 QAM cluster centers. The Outstar may then be trained. About 8000 iterations later, *Train Outstar* may be turned off.

Step 6 Select File-Close on the Simulation Run window.

9. Improving Viterbi Decoder Soft Decisions in a Pulse Jamming Environment (refer to 4.9)

Step 1 We initialize the network weights to values previously determined by training the network to learn an identity mapping. Thus the initial transfer function for the neural network will be linear over a range of signal values typical of a no-jammer scenario. This transfer function is similar to a conventional soft decision metric. To do this: cp /spwsys/pool/tbill_3_all/viterbi_id_10nodes /spwsys/pool/nncss_all/viterbi3.net

Step 2 Open the SPW simulation model entitled bp_viterbi(24).system.

Step 3 Run the simulation for 70K iterations. You will observe the neural network soft decision metric adapt from its initial form to a form similar to the theoretical optimum.

Step 5 Select File-Close on the Simulation Run window.

10. Demodulation of QPSK Over a Non-Linear, Dispersive Channel Using a Fully Recurrent Network (refer to 4.10)

This demonstration displays the results of a trained Recurrent Network to demodulate a QPSK signal which has been transmitted over a non-linear, dispersive, AWGN channel. The actual training of the network is time-consuming and is not appropriate for an ISL demonstration. To execute the ISL demonstration:

Step 1 Since this demonstration displays results for a *previously* trained network, we must begin with the weights which resulted from this earlier training: cp /spwsys/pool/tbill_3_all/qpsk6new.net /spwsys/pool/nncss_all/qpsk6new.net

Step 2 Open the SPW simulation model entitled rc_qpsk(3).system.

Step 3 Make sure that learning and *forced* is off by changing the value of the constant feeding into the *train* and *forced* input to the rcnet block to 0.0.

Step 4 Run the simulation for 20K iterations. The in-phase and quadrature components produced by the neural network (shown in a constellation diagram) are thresholded to produce the demodulated QPSK symbol.

Step 5 Select File-Close on the Simulation Run window.

To see that the neural network can indeed be trained to demodulate a QPSK signal which has been transmitted over a non-linear, dispersive, AWGN channel, this simulation can be executed in a non-ISL mode with the following modifications:

Step 1 Make sure that weights are initialized from random values instead of from the stopping point of a previous execution of this system: rm /spwsys/pool/nncss_all/qpsk6new.net

Step 2 If it is not already on, turn learning on and *forced* on by changing the value of the constant feeding into the *train* input and *forced* input to the rcnet block to 1.0.

Step 3 To expedite the simulation, select and cut all of the ISL blocks in the ISL Output portion of the system diagram.

Step 4 In the Network and Output Control portion of the system diagram, edit the parameter on the Unit Step block which feeds into the Inverter and *wait* connector. Change its value to 950000. This will cause signal files to be written to disk only after simulation iteration 950000 has been reached, thus writing less to disk.

Step 5 Run the simulation for 1000000 iterations. This may take about several hours to complete.

Step 6 The Recurrent Network has been trained with forced learning. It now requres further training with unforced learning. Turn learning on and *forced* off.

Step 7 In the Network and Output Control portion of the system diagram, edit the parameter on the Unit Step block which feeds into the Inverter and *wait* connector. Change its value to 950000. This will cause signal files to be written to disk only after simulation iteration 950000 has been reached, thus writing less to disk.

Step 8 Run the simulation for 1000000 iterations. This may take about several hours to complete.

Step 9 After completion, press SigCalc on the Simulation window to view resulting signals.

Step 10 Select File-Close on the Simulation Run window.

*U.S. GOVERNMENT PRINTING OFFICE: 1994-510-117-50036

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.